VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE MŘÍŽKOVÉHO ROZLOŽENÍ VE ZOBRAZOVACÍM STROJI CSS

GRID LAYOUT IMPLEMENTATION IN CSS RENDERING ENGINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ NOVÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Novák Ondřej**
Program: Informační technologie
Název: **Implementace mřížkového rozložení ve zobrazovacím stroji CSS**
Grid Layout Implementation in a CSS Rendering Engine
Kategorie: Web

Zadání:

1. Seznamte se s projektem experimentálního zobrazovacího stroje CSSBox a jeho architekturou.
2. Prostudujte nové způsoby rozložení obsahu pomocí jazyka CSS jako např. grid layout a flexbox.
3. Navrhněte způsob rozšíření knihovny CSSBox o nové způsoby rozložení obsahu.
4. Po dohodě s vedoucím implementujte podporu rozložení "Grid" ve zobrazovacím stroji.
5. Proveďte testování vytvořeného rozšíření pomocí vhodné testovací sady.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Dokumentace projektu CSSBox: http://cssbox.sourceforge.net/documentation.php
- Michálek, M.: Vzhůru do CSS3, e-kniha, 2015, https://www.vzhurudolu.cz/ebook-css3
- Andrew, R.: Get Ready for CSS Grid Layout, A Book Apart, 2016, https://abookapart.com/products/get-ready-for-css-grid-layout

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 30. října 2018

Abstrakt

Cílem této bakalářské práce je prostudovat architekturu experimentálního zobrazovacího stroje CSSBox a navrhnout způsob rozšíření knihovny o nové možnosti mřížkového rozložení obsahu. Nejdříve je uveden úvod do problematiky, po kterém následuje samotný návrh řešení. Navržená architektura je implementována a otestována. V závěru práce jsou zhodnoceny výsledky a naznačeny další možnosti rozvoje.

Abstract

The goal of this thesis is to study the architecture of the CSSBox experimental rendering engine and propose a way to expand library with new options of grid layout content. The opening chapters contain an overview of problematics and subsequently a solution is proposed. The proposed architecture is implemented and tested. The conclusion is dedicated to evaluation of results and options futher development are outlined.

Klíčová slova

Mřížkové rozložení, CSSBox, CSS, HTML, XHTML, DOM, Java

Keywords

Grid layout, CSSBox, CSS, HTML, XHTML, DOM, Java

Citace

NOVÁK, Ondřej. *Implementace mřížkového rozložení ve zobrazovacím stroji CSS*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Implementace mřížkového rozložení ve zobrazovacím stroji CSS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Novák
15. května 2019

Poděkování

Chtěl bych poděkovat svému vedoucímu panu Ing. Radku Burgetovi Ph.D. za vedení této bakalářské práce, za poskytnutí odborné pomoci a za rady při jejím zpracování.

Obsah

Kapitola 1

Úvod

V poslední době využívání webových prohlížečů velice stoupá, velkou oblíbenost našly také mobilní webové prohlížeče. Důležité tedy je, věnovat se vývoji webových nástrojů, které zjednodušují tvorbu webů, vzhledem k tomu, že díky mobilním telefonům přibylo mnoho různých rozlišení displejů. Z toho plyne, že je kladen větší důraz přizpůsobení webových stránek vůči jednotlivým displejům. Na tento problém reagoval jazyk CSS, který s sebou přinesl tzv. dotazy na média, které umožňují aplikovat různé typy rozložení na různá rozlišení. V pozdější fázi vývoje tento jazyk přispěl i novými typy responzivního rozložení webových stránek, jako je například mřížkové rozložení (Grid Layout) nebo Flexible Box layout. A právě tímto novým typem mřížkového rozložení se bude zabývat i tato bakalářská práce.

Cílem této bakalářské práce je seznámení se s experimentálním zobrazovacím strojem CSSBox, který je využíván pro potřeby zpracování a vyhodnocování webových stránek, a jeho následné rozšíření o nové prvky mřížkového rozložení.

V kapitole 2 je čtenář seznámen s teoretickou částí a pojmy, se kterými je zde pracováno. V závěru této kapitoly jsou popsány typy rozložení, které se aktuálně mohou využít pro strukturu webových stránek. Se závěrem této kapitoly souvisí také kapitola 3, která je věnována podrobnému popisu Grid Layoutu.

Kapitola 4 popisuje zobrazovací stroj CSSBox, kde je řešena jeho aktuální struktura či různá omezení. V podkapitole 4.4 je rozebrána knihovna jStyleParser, která spolupracuje právě s knihovnou CSSBox. Následuje kapitola 5, která navazuje na předchozí kapitolu, a zabývá se návrhem rozšíření pro knihovnu CSSBox.

Kapitola 6 se zabývá samotnou implementací Grid Layoutu. Je zde zachyceno provedení řešení, mezi které patří struktura implementace, algoritmy a další detaily programové realizace. Následně je prováděno testování, které je předmětem kapitoly 7. V této kapitole je zahrnuto i diskutování budoucího vývoje Grid Layoutu a možná rozšíření práce.

Kapitola 2

Teoretická část

Tato kapitola se zabývá popisem informací, které jsou spojeny s touto prací. Předmětem této kapitoly je studium jazyků HTML a XHTML, DOM modelu pro HTML dokument a CSS. V poslední části této kapitoly jsou krátce popsány typy rozložení, které se využívají pro webové stránky, s výjimkou Grid Layoutu, který je podrobněji popsán v kapitole 3.

2.1 Jazyk HTML

HTML [20] (Hyper-textový značkovací jazyk) je standardní značkovací jazyk pro tvorbu webových stránek v systému WWW[1]. Je to jazyk pro popis struktury webových stránek pomocí tzv. tagů[2], kde každý tag reprezentuje jeden příkaz, a tyto příkazy jsou uzavřeny do ostrých závorek (<, >). Obvykle jsou tyto tagy párové, kde startovní tag značí začátek daného příkazu, a koncový tag udává konec příkazu. Koncový tag je zapsán stejně jako startovní, akorát se zde navíc vyskytuje znak lomítka. Webové prohlížeče tyto tagy nezobrazují, ale využívají je pro vykreslování obsahu stránek. Tagy mohou navíc obsahovat i atributy, které poskytují další informace o daném tagu. Atributy se zapisují ve tvaru: jméno="hodnota". Jazyk HTML také umožňuje propojovat jednotlivé stránky pomocí hypertextových odkazů.

Nejdůležitější tag celého dokumentu je `<html>`, do kterého je vložena celá struktura webové stránky. Uvnitř se pak musí povinně vyskytovat tagy `<head>` a `<body>`. V tagu `<head>` jsou obsaženy meta informace o dané stránce a dají se zde připojovat i styly CSS nebo JavaScript soubory. Navíc je zde uveden tag `<title>`, který specifikuje název dokumentu. Tag `<body>` je využit pro obsah dokumentu. Všechen obsah, co je zde zapsán, dokáží webové prohlížeče zobrazit. Dalším důležitým tagem je tag `<!DOCTYPE>` reprezentující typ dokumentu a pomáhá webovým prohlížečům správně zobrazit webové stránky. Tento tag se musí objevit pouze jednou a je umístěn v horní části dokumentu, ještě před tagem `<html>`.

2.2 Jazyk XHTML

XHTML [7] je značkovací jazyk pro tvorbu webových dokumentů, kde X na začátku znamená eXtensible neboli rozšiřitelný. Byl vyvinut mezinárodním konsorciem W3C[3], jako

[1] World Wide Web – Celosvětová síť
[2] Tag – značka
[3] World Wide Web Consortium

náhrada staršího jazyka HTML, tak aby splňoval standard XML[4]. XHTML jako takové nepřináší žádné nové možnosti, ale pouze omezení. V současnosti se XHTML vyskytuje ve třech verzích. Verze 1.1 a verze 1.0, která se rozlišuje na Transitional a Strict. Oproti HTML se například změnila hlavička, kterou požaduje XML. Nepárové tagy musí být ukončeny, všechny tagy a atributy musí být zapsány malými písmeny a atributy nesmějí být prázdné. Dnes jsou jazyky HTML a XHTML spojeny do standardu HTML5.

2.3 DOM model

DOM [16] (Document Object Model) je programovací API[5] pro dokumenty HTML a XML. Je implementován ve velké škále programovacích jazyků (například Java, PHP, Perl, Python a další.). Definuje logickou strukturu dokumentů, která se velmi podobá stromu. Díky tomuto rozhraní lze pracovat se samotnou webovou stránkou. DOM umožňuje přistupovat k jednotlivým objektům dokumentu a poskytuje možnost pracovat s nimi. Objektový model dokumentu však neuvádí, že by se měly dokumenty implementovat jako strom, a ani neuvádí, jakým způsobem se budou implementovat vztahy mezi objekty. Jinak řečeno, objektový model specifikuje logický model pro programovací rozhraní a tento logický model může být implementován jakýmkoliv způsobem, který daná implementace považuje za výhodný. [19]

Jednou z důležitých vlastností DOM modelů je strukturální izomorfismus. Což znamená, že pokud se pro vytvoření reprezentace stejného dokumentu použijí nějaké dvě implementace DOM, vytvoří se stejný model struktury s přesně stejnými objekty a vztahy. Díky stromové struktuře dokumentu jsou možné rozlišovat typy elementů, jako jsou nadřazené, podřazené a rovnocenné elementy. K dokumentu může DOM přistupovat prostřednictvím několika objektů:

- Objekt Element - reprezentující elementy
- Objekt Attr - reprezentující atributy
- Objekt Comment - reprezentující komentáře
- Objekt Text - reprezentující text

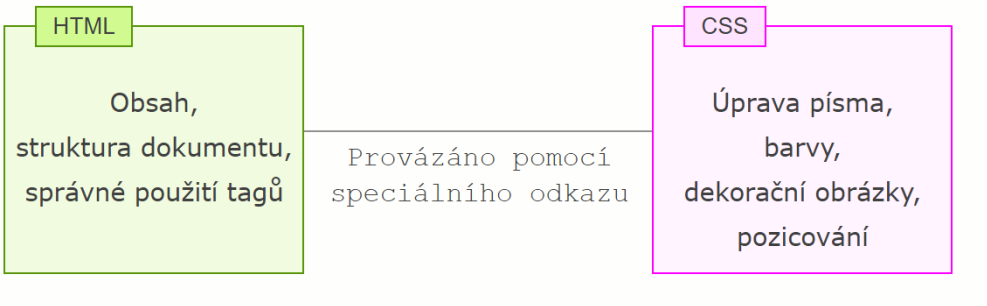
2.4 CSS

CSS [18] (Kaskádové styly) se označují jako deklarativní jazyk, který se v současné době využívá pro popis prezentace webových stránek, včetně různých barev, písem a typů rozložení stránek. Umožňuje přizpůsobit prezentaci různým typům zařízení, jako jsou například malé či velké obrazovky. Styly CSS jsou nezávislé na HTML a dají se použít s jakýmkoliv značkovacím jazykem založeným na XML. Dají se zapsat přímo do HTML tagů, kde se jim přidá atribut `style=""`, nebo do HTML hlavičky pomocí tagu `<style>`. Nejpoužívanějším způsobem zápisu stylů je však externí soubor, který se pomocí tagu `<link>` připojí k HTML hlavičce. Vztah mezi HTML a CSS je vyobrazen na obrázku 2.1.

Přiřazení specifických vlastností k HTML elementům se provádí pomocí selektorů. Za selektory se dají považovat například elementy `p, h1, body`. Dále sem patří selektory tříd,

[4] eXtensible Markup Language – rozšiřitelný značkovací jazyk
[5] Application Programming Interface – rozhraní pro programování aplikací

které vybírají elementy na základě definované třídy, příklad zápisu může vypadat takto: `.nadpis`. Nebo selektory definované na základě identifikátorů, které se dají zapsat například jako: `#odstavec`.



Obrázek 2.1: Vztah mezi HTML a CSS. [15]

Výhody

CSS poskytuje mnohem rozsáhlejší možnosti formátování než prvky HTML určené k ovládání vzhledu. Každému prvku lze nastavit jeho přesné rozměry, které nemusí být jen pomocí pixelů, ale můžou být i pomocí relativních jednotek. Prvek lze formátovat i jako blok textu, kterému můžeme nastavit vzdálenost od jiného elementu nebo vzdálenost od okrajů stránky. V jazyce HTML by se toto odsazení muselo řešit například pomocí tabulek, ale v CSS stačí využít vlastností `margin` či `padding`.

Stránky, které jsou formátované pomocí CSS, jsou snazší na udržování webové reprezentace. Většinou jsou styly uloženy v jediném souboru CSS, ale ne vždy to tak musí být, tudíž pokud je potřeba provést nějaká grafická úprava, nemusí se zdlouhavě procházet HTML soubor, ale stačí pozměnit pouze vlastnost v CSS souboru a úprava se promítne pro všechny zvolené prvky.

U stránek, které využívají CSS formátování, je v HTML souboru uložen pouze obsah a jeho struktura, takže jeho velikost je podstatně menší. Připojený soubor CSS si prohlížeče stáhnou pouze jednou a následně si ho uloží. Co z toho plyne? Stránky o menší velikosti se uživatelům nahrávají mnohem rychleji. Sníží se i zatížení serveru díky menším souborům, které se přenáší k uživateli. [13]

Nevýhody

CSS s sebou nese i jisté nevýhody. Jedna z nevýhod je ne vždy dokonalá podpora ve webových prohlížečích. Ne vždy dokáží prohlížeče zobrazit zamýšlený obsah, tudíž se musí dané konstrukce řešit například pomocí vizuálních prvků jazyka HTML, nebo specifikovat, jak se daná konstrukce v konkrétním prohlížeči má zobrazit. [13]

2.4.1 CSS 1

Je to první specifikace CSS [6], která se stala oficiálním doporučením konsorcia W3C. Byla vydána v prosinci roku 1996. Zavedla syntaxi zápisu, která se využívá až do dnes, dále pak styl jak vybírat elementy přes selektory a hodnoty s jejich příslušnými jednotkami. Mezi možnosti CSS 1 patří například podpora:

- vlastnosti písma a textu
- barva textu a pozadí
- zarovnání textu, obrázků a tabulek
- ohraničení a pozicování většiny elementů
- Margin a padding vlastnosti

2.4.2 CSS 2

Tato verze byla vydána v květnu roku 1998. Je to nadmnožina specifikace CSS 1. Přinesla řadu nových možností jako je absolutní, relativní, pevné umístění elementů, z-index, podpora zvukových stylů, nové vlastnosti písma. A koncept typů médií, kde se pomocí příkazu `@media` daly definovat různé styly pro různé typy médií. Bohužel nedostaly moc velkou podporu, tak byly později upraveny a rozšířeny v CSS 3.

Za zmínku také stojí verze CSS 2.1., která vlastně nepřinesla nic nového, ale pouze opravuje chyby ve verzi CSS 2. [6]

2.4.3 CSS 3

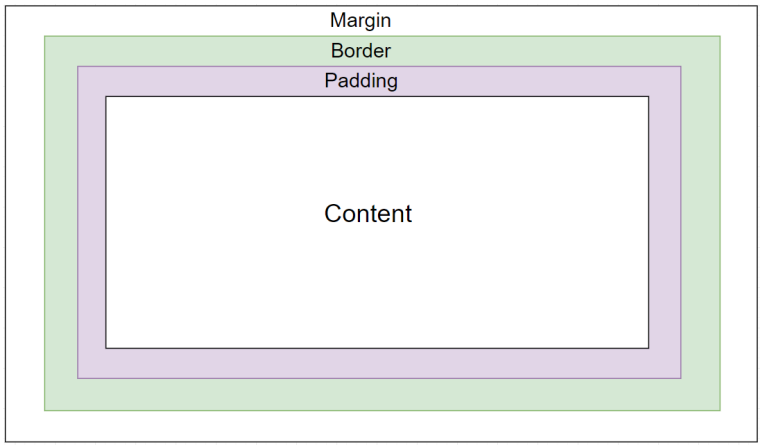
První funkce CSS 3 byly převážně prezentačního charakteru a řešily problémy, které se dlouhé roky programátoři snažili obcházet. Byly to například používání vlastních písem, kulatých rohů, stínové efekty pod elementy a písmeny. Následovaly je nové selektory, které ulehčují procházení dokumentu za účelem stylování, a také dynamické efekty, mezi které patří dvou a tří dimenzionální přechody a přechodové animace.

Velký rozvoj stylů CSS také přinesly tzv. media queries[6]. Dají se označit za speciální syntaktické konstrukce, které umožňují stylovat stránku na základě rozměrů zobrazovacího média a schopností webového prohlížeče. Media queries představují logický výrok. Pokud je výrok pravdivý, všechny styly se aplikují pro tento výrok, v opačném případě je aplikace těchto stylů přeskočena. [5]

2.5 CSS Box Model

CSS Box model [12] je základem rozložení na webových stránkách. Každý element je reprezentován jako obdélník se svým obsahem (content), který je určen svojí šířkou a výškou, dále jsou pak na obsahu nalepeny ve vrstvách vlastnosti padding (vnitřní okraj), border (rámeček) a margin (vnější okraj), které jsou také reprezentovány jako obdélníky. Na obrázku 2.2 je znázorněn CSS Box model, včetně jeho všech vrstev. Jakmile prohlížeč vykreslí webovou stránku, je schopen najít, jaké styly CSS jsou aplikovány na CSS Box model, a díky tomu je schopen zjistit výslednou velikost a umístění.

[6] media queries – dotazy na média



Obrázek 2.2: CSS Box model.

2.6 Typy rozložení

Co to vlastně rozložení [14] (layout) je? Dá se popsat jako způsob polohování oblastí stránky s různým typem obsahu. Layout lze rozdělit na dva základní typy:

- podle počtu sloupců – jednosloupcový, dvojsloupcový, trojsloupcový
- podle nastavení šířky stránky
  - fixní – pracuje s pevnou šířkou stránky, obvykle se využívá šířky 960px
  - fluidní – pracuje s proměnnou šířkou stránky, šířka se mění dle aktuální šířky okna. Oproti fixnímu rozložení nepoužívá pevné jednotky, ale využívá relativní jednotky (procenta, em). Obvykle se využívá šířky 90%.
  - responzivní – přizpůsobuje se šířce okna a zaručuje, že stránka bude správně zobrazena na nejrůznějších typech zařízení (monitory, tablety, mobily)

Tabulkový layout

Tabulkový layout [14] je jeden ze základních typů rozložení. Webová stránka je rozložena do tabulky, na kterou stačí obyčejné HTML bez realizace vlastností CSS. Kód je velice jednoduchý a přehledný, tudíž je funkční ve všech webových prohlížečích. Bohužel vše má i své temné stránky. U tabulkového rozložení spočívá největší nevýhoda v načítání celé tabulky najednou, což znamená, že obsah se zobrazí až po načtení celé tabulky. Dalšími nevýhodami jsou: špatná přenositelnost mezi zařízeními (může se vyskytovat horizontální posuvník) a omezené formátování buněk tabulky.

Dnes se již toto rozložení moc nevyužívá a je nahrazeno modernějšími způsoby rozložení webových stránek.

Rámcový layout

Rámcový layout [4] při tvorbě využívá tzv. rámců (frames). Seskupuje obsah několika HTML stránek pomocí speciální rámcové stránky. Základem je tedy rámcová stránka, která definuje rozložení obsahu stránek do určitých rámcových oblastí.

Za takové typické rámcové rozložení lze považovat stránku, která má navigační menu v užším rámci, který je nejčastěji umístěn v levé části nebo horní části stránky. A při kliknutí na nějakou položku v navigačním menu se celý obsah zobrazí v hlavním rámci.

Mezi výhody se dá zařadit navigační menu, které je společné pro všechny stránky, a zůstává na svém místě, tudíž se k němu nemusí zdlouhavě rolovat. Dále pak jednoduchost a efektivnost rozložení obsahu. Mezi nevýhody patří pomalé načítání stránky a nevhodnost pro vyhledávače, které uživateli nemusí vždy nabídnout celou rámcovou strukturu, ale nabídnou pouze obsah stránky.

Dnes se již toto rozložení také moc nevyužívá, ale stále se najdou tací příznivci, kteří toto rozložení využívají.

Blokový layout

Blokový layout [4] využívá pozicování[7] při rozložení stránky. Pro účely pozicování byly vytvořeny dva speciální tagy `<div>` a `<span>`. Jsou to univerzální tagy, které lze formátovat pomocí jakékoliv vlastnosti CSS.

Tag `<div>` je blokový a používá se pro rozložení stránky. Lze do něj zapisovat další vnořené tagy. Roztáhne se po celé dostupné šířce, tudíž vytváří před sebou a za sebou konec řádku, ale lze mu i libovolně nastavovat velikost. Tento tag může být použit i pro obalení nějaké části stránky, které se můžou nastavit stejné vlastnosti CSS.

Tag `<span>` je řádkový a používá se pro formátování nějaké části textu. Nemůže v sobě obsahovat další vnořené blokové tagy.

Mezi výhody lze zařadit rychlé načítání stránky a pokročilejší možnost stylování vzhledu stránky.

Flexible Box Layout

Flexible Box Layout [11] je další typ nového responzivního rozložení, který definuje jednorozměrný systém uspořádání obsahu. Funguje na podobném principu jako Blokový layout, s tím rozdílem, že je více zaměřený na zarovnávání a rozdělování obsahu. Obecně se spíše využívá na rozložení určité části obsahu. Flexible Box s sebou přinesl i novou hodnotu pro vlastnost display, `display:flex`. Díky této hodnotě vznikne flex kontejner, který lze orientovat řádkově i sloupcově. Přímí potomci kontejneru se nazývají flex položky, v kontejneru se umisťují ihned za sebou a dle určitých vlastností se dají v kontejneru různě pozicovat.

[7] https://www.jakpsatweb.cz/css/css-pozicovani.html

Kapitola 3

Grid Layout

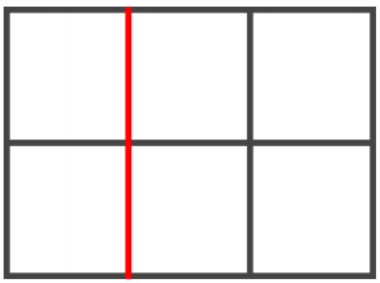
Grid Layout [8] (mřížkové rozložení) je nový modul CSS, který definuje dvojrozměrný systém uspořádání mřížky, a řadí se mezi typy nového responzivního rozložení webové stránky. Mřížku tvoří množina protínajících se vertikálních a horizontálních linek, které rozdělují mřížku do oblastí, mezi které lze umisťovat položky mřížky (tzv. grid items). Tyto položky představují oblast celé mřížky.

3.1 Základní terminologie

V této kapitole je probírána základní terminologie. Vyskytují se zde pojmy, které ve spojení s Grid Layoutem jsou využívány. Mezi tyto nejdůležitější pojmy patří linky mřížky, stopy mřížky, buňky mřížky, oblasti mřížky a tzv. žlaby mřížky.

Grid Lines

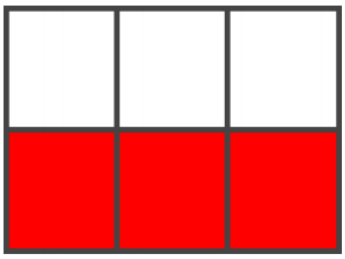
Grid Lines [17] (linky mřížky) se dělí na dva typy linek, a to buď vertikální, nebo horizontální a vytvářejí celou mřížku. Názorná ukázka linky v mřížce je znázorněna na obrázku 3.1. Linka existuje na obou stranách řádku i sloupce. Linky mohou být označeny pomocí číselného indexu, který linky začíná indexovat od čísla jedna, nebo se dají slovně pojmenovat, například podle jejich funkce v mřížce.



Obrázek 3.1: Vertikální linka mřížky s indexem 2. [1]

Grid Track

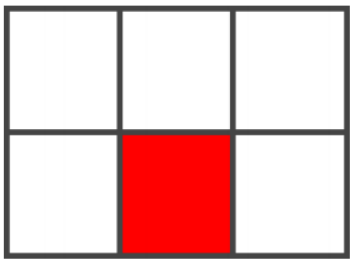
Grid Track [1] (stopa mřížky) je obecný výraz pro sloupec nebo řádek uvnitř mřížky, jinak řečeno, je to mezera mezi dvěma sousedními linkami. Může být také vertikální nebo horizontální. Příklad stopy mřížky je znázorněn na obrázku 3.2. Každá stopa mřížky má svojí funkci pro dimenzování, která udává, jak vysoký nebo široký může být řádek či sloupec, nebo-li jak jsou od sebe vzdáleny linky. Sousední stopy mohou být od sebe odděleny tzv. okapy (viz. strana 12).



Obrázek 3.2: Stopa mřížky mezi linkami 2 a 3. [1]

Grid Cell

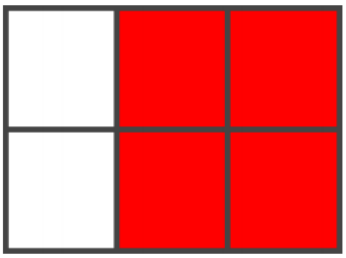
Grid Cell [17] (buňka mřížky) je prostor mezi čtyřmi linkami v mřížce. Je to nejmenší jednotka mřížky, na kterou se lze odkazovat při umisťování položek mřížky. Zobrazení buňky v mřížce je uvedeno na obrázku 3.3. Buňka mřížky se dá konceptuálně považovat za buňku tabulky.



Obrázek 3.3: Buňka mřížky mezi horizontálními linkami 2 a 3 a vertikálními linkami 2 a 3. [1]

Grid Area

Grid Area [17] (oblast mřížky) je logický prostor, který slouží k rozložení jedné nebo více položek mřížky. Oblast mřížky se skládá z jedné nebo více sousedních buněk mřížky. Je to vazba čtyř linek, jedna na každé straně oblasti, a zúčastňuje se na dimenzování stop v mřížce, které protíná. Názorný příklad, jak by taková oblast mohla vypadat, je vyobrazen na obrázku 3.4.



Obrázek 3.4: Oblast mřížky mezi horizontálními linkami 1 a 3 a vertikálními linkami 2 a 4. [1]

Oblasti mřížky se dají také jednoduše pojmenovat. Díky explicitnímu pojmenování lze snadně navrhovat vzhled stránky. Prostřednictvím příkazu `grid-template-areas` se nadefinují jednotlivé oblasti. Na obrázku 3.5 lze vidět jednoduché rozložení stránky, které se skládá ze tří sloupečků a tří řádků. Celý první řádek je přiřazen do oblasti pro záhlaví. Poslední řádek je rezervován pro patičku. Prostřední řádek je rozdělen na menu a obsah, kde menu zabírá pouze první sloupec a obsah zbývající dva.

```
grid-template-areas: "header header header"
                     "menu content content"
                     "footer footer footer";
```

Obrázek 3.5: Jednoduchý vzhled stránky definovaný pomocí pojmenovaných oblastí.

Grid Gutters

Grid Gutters [17] (tzv. žlaby mřížky) definují mezery mezi řádky a sloupci mřížky v grid kontejneru pokud jsou zadány. Jestliže nejsou nastaveny, tak se chovají jako by tvořily mezeru o velikosti nula. Efekt těchto vlastností vypadá, jako by ovlivněné linky mřížky získaly tloušťku. Tyto žlaby se objevují pouze mezi stopami implicitní mřížky (viz. kapitola 3.2.2), ale před první a polední stopou žádné žlaby nejsou.

Pro účely dimenzování je každá ovlivněná linka považována za extra prázdnou stopu mřížky s pevnou velikostí, která jí je zadána. To znamená, že pokud bude grid item umístěn přes dvě a více stop mřížky, musí se do jeho velikosti připočítat i veškeré linky, mající definovanou velikost.

3.2 Grid Container

Grid container [17] (kontejner mřížky) zavádí dvě nové hodnoty pro vlastnost CSS display. První z nich je `display:grid`, tato hodnota způsobuje, že element generuje grid kontejner, který je považován za blokový, když je umístěn ve flow-layout[1]. Druhá hodnota je `display:inline-grid`, která způsobuje, že element generuje grid kontejner, který je považován za inline, když je umístěn ve flow-layout.

[1] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flow_Layout

Grid kontejner vytváří nový kontext formátování mřížky pro svůj obsah. Je to stejné jako vytváření kontextu blokového formátování s tím rozdílem, že místo blokového rozložení se použije mřížkové rozložení. Obsah grid kontejneru se rozkládá do mřížky a linky mřížky tvoří hranice pro každou položku uvnitř mřížky a to včetně vnořených bloků, které odpovídají dané položce.

Grid kontejnery ale mají odlišné vlastnosti než blokové, tudíž se nedají považovat za sobě rovné, a tak některé vlastnosti, které byly navrženy s předpokladem blokového rozložení, nemůžou být aplikovány v kontextu mřížkového rozložení. Mezi tyto vlastnosti se považují:

- `float` a `clear` – tyto vlastnosti nemají žádný efekt na položku v mřížce. Nicméně `float` stále může ovlivňovat vypočítanou hodnotu `display`.
- `vertical-align` – vertikální zarovnání, které také nemá žádný efekt na položku v mřížce

Za zmínku také stojí to, že pokud je element plovoucí nebo absolutně umístěný a jeho hodnota display je nastavena na `display:inline-grid`, tak je tato hodnota přepočítána na `display:grid`.

3.2.1 Explicitní mřížka

Explicitní mřížka [17] grid kontejneru je definována pomocí následujících tří vlastností: `gridtemplate-rows`, `grid-template-columns` a `grid-template-areas`. Jak veliká bude explicitní mřížka určuje počet sloupců a řádků. které definuje vlastnost `grid-template-areas` a velikosti řádků či sloupců, které udávají vlastnosti `grid-template-rows` a `grid-template-columns`. Vlastnost `grid-template-areas` určuje pouze pojmenované oblasti mřížky, které nejsou přiřazeny žádnému konkrétnímu grid itemu, ti se ale na tuto oblast mohou odkazovat. Pokud ale není nastavena žádná z těchto vlastností, tak explicitní mřížka stále existuje, což znamená, že obsahuje jednu linku, jak v horizontální, tak i ve vertikální ose. Na obrázku 3.6 je znázorněno vytvoření explicitní mřížky.

Může nastat i případ, kdy grid item bude umístěn mimo explicitní mřížku. V tomto případě se explicitní mřížka rozšíří požadovaný počet implicitních stop mřížky.

```
grid-template-rows: 50px 200px 40px;
grid-template-columns: 100px 200px 80px;
grid-template-areas: "header header header"
                     "menu content content"
                     "footer footer footer";
```

Obrázek 3.6: Příklad vytvoření explicitní mřížky.

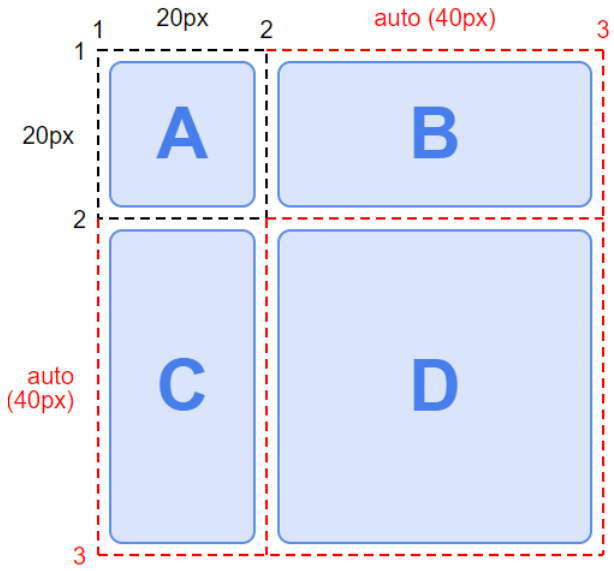
Za zmínku také stojí funkce `repeat()` a `minmax()`. Funkci `repeat()` lze použít jako alternativu místo zdlouhavého zápisu velikostí řádků či sloupců. Například při vytváření velké mřížky můžeme nadefinovat sloupce jako `repeat(10, 150px)`, což znamená, že se vytvoří deset sloupců o velikosti 150px. Funkce `minmax()` vyjadřuje ohraničení velikosti stopy mřížky, což lze chápat, jako nejmenší a největší velikost stopy. Například zápis `minmax(50px, 150px)` znamená, že stopa může být velká v rozmezí od 50px do 150px.

V poslední řadě můžeme celé vytváření explicitní mřížky shrnout do dvou vlastností, `grid` a `grid-template`. Tyto vlastnosti nepřináší nic nového, ale jsou to pouze zkratky

předchozích vlastností. Jsou to vesměs totožné zkratky, s tím rozdílem, že `grid` resetuje i vlastnosti implicitní mřížky, zatímco `grid-template` je ponechává beze změny.

3.2.2 Implicitní mřížka

Implicitní mřížka [17] je vesměs aktivní po celou dobu Grid Layoutu. Znamená to tedy, jakmile se grid kontejneru specifikuje vlastnost `display:grid`, implicitní mřížka je automaticky vytvořena. Pokud je nějaký grid item umístěný mimo hranice explicitní mřížky, grid kontejner ihned generuje dodatečné linky mřížky, které se přidají do kontejneru, a tím vytváří potřebné implicitní stopy mřížky. Tyto nově přidané linky spolu s explicitní mřížkou tvoří dohromady implicitní mřížku. Na obrázku 3.7 lze vidět příklad spojení explicitní mřížky s implicitně vytvořenými stopami.



Obrázek 3.7: Znázornění 2x2 mřížky s jednou explicitní buňkou mřížky **A** o rozměrech 20x20 pixelů a třemi dalšími buňkami **B**, **C** a **D**, které se implicitně vytvořili pro další grid itemy. [17]

Nyní se zaměříme na vlastnosti, které jsou spojeny s implicitní mřížkou. Mezi tyto vlastnosti patří: `grid-gap`, `grid-auto-rows`, `grid-auto-columns` a `grid-auto-flow`.

Vlastnost `grid-gap` lze definovat pouze v implicitní mřížce nebo ve spojení s explicitní mřížkou. Na pohled vytváří mezery mezi grid itemy. Lze jí definovat pouze pevné jednotky (například pixely) nebo procenta. Je to nadmnožina CSS vlastností `grid-row-gap` a `grid-column-gap`, které akorát blíže specifikují, zda se mají vytvořit mezery pouze pro řádky či sloupce.

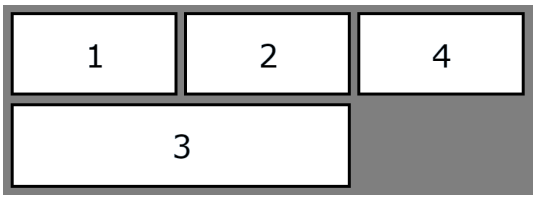
Vlastnosti `grid-auto-rows` a `grid-auto-columns` vytvářejí nové řádky nebo sloupce pro grid itemy jestliže je potřeba. Pokud je item umístěn mimo explicitní mřížku jsou pro něj vytvořeny automatické řádky či sloupce. Tyto vlastnosti se také využívají při automatickém umisťování itemů, když se item už nevejde na řádek či sloupec, jsou pro něj vytvořeny nové.

Vlastnost `grid-auto-flow` se stará o automatické umisťování grid itemů, kterým nebyla specifikovaná explicitní pozice umístění. Znamená to tedy, že se itemy umístí automaticky do volného místa v mřížce podle zvolených hodnot:

- row – grid itemy jsou postupně umisťovány do volných prostorů v řádcích a pokud už nezbývá volný prostor, automaticky přidá nový řádek
- column – grid itemy jsou postupně umisťovány do volných prostorů ve sloupcích a pokud už nezbývá volný prostor, automaticky přidá nový sloupec
- dense – speciální typ umisťování grid itemů, snaží se vyplňovat prázdné prostory dříve v mřížce, pokud se později objeví menší itemy. Na obrázku 3.8 lze vidět umístění itemů podle hodnoty dense.

Nakonec stejně jako u explicitní mřížky můžeme celé vytváření shrnout do jedné vlastnosti `grid`, která se využívá pro zkrácení dlouhých příkazů.

Grid kontejner může obsahovat i několik vlastností pro zarovnání itemů. Vlastnost `align-items` zarovnává všechny itemy ve vertikální ose neboli ve sloupcích mřížky. Vlastnost `justify-items` zarovnává všechny itemy v horizontální ose neboli v řádcích mřížky. Vlastnost `align-content` zarovnává všechny stopy mřížky ve vertikální ose. Vlastnost `justify-content` zarovnává všechny stopy mřížky v horizontální ose.



Obrázek 3.8: Ukázka řazení grid itemů podle vlastnosti `grid-auto-flow:dense`. Item 4 se umístil za item 2, protože jeho velikost odpovídá volnému prostoru, lze tedy tento item využít pro vyplnění volného prostoru v mřížce.

3.3 Grid Item

Grid item [17] (položka mřížky) se dá považovat za box, který reprezentuje in-flow[2] obsah pro svůj grid kontejner. Každý in-flow potomek grid kontejneru se stává položkou mřížky. Pokud by položka obsahovala nějaké vnořené elementy, tak jejich maximální obsah je roven obsahu položky mřížky. Za vnořené elementy se dají považovat blokové a inline elementy. Nicméně pokud se použije jejich kombinace, tak jsou inline elementy obaleny do tzv. annonymous[3] boxů.

V poslední řadě není na škodu také zmínit, že pokud se do grid itemu zapíší pouze bílé znaky, ty nejsou zobrazeny, protože mají nastavené chování, jako by jim byla nastavena vlastnost `display:none`. Avšak grid item jako takový je normálně vykreslen.

3.3.1 Umisťování položek do mřížky

Každý grid item je spojen se svojí oblastí v mřížce. Po umístění itemu do mřížky se z dané oblasti stane blok, který určuje jeho polohu. Grid item lze samozřejmě umístit i do buňky mřížky, ze které se následně stane také blok.

[2] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flow_Layout/In_Flow_and_Out_of_Flow

[3] annonymous box – je takový box, který se nedá řešit pomocí selektorů CSS a všechny jeho vlastnosti mají výchozí hodnoty, s výjimkou vlastnosti `display`

Lokace grid itemu je definována svými souřadnicemi, které se vztahují s jednotlivým linkám mřížky, nebo svým rozpětím. Vlastnosti `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end` a jejich zkrácené zápisy `grid-column`, `grid-row` slouží pro umístění grid itemu. Na obrázku 3.9 lze vidět příklad umístěného itemu do určité oblasti. Vlastnosti `grid-column-start` a `grid-row-start` určují počáteční linky mřížky, od kterých budou itemy umístěné. Vlastnosti `grid-column-end` a `grid-row-end` zase určují koncové linky mřížky, po které budou itemy umístěné.

Rozpětí grid itemu v mřížce udává o kolik linek mřížky se item roztáhne. Rozpětí je vždy určité, s počáteční hodnotou jedna, pokud není jinak specifikována. Může být zadáno ve dvou typech pro řádek nebo sloupec. První typ definuje jak se item roztáhne od svého počátku a druhý typ definuje roztažení itemu od svého konce. Například zápis `span 2` udává roztažení itemu o dvě mřížkové linky.

```
.item {
        grid-column-start: 1;
        grid-column-end: 3;
        grid-row-start: 1;
        grid-row-end: 2;
}
```

Obrázek 3.9: Příklad umístění grid itemu do oblasti mřížky.

Grid item lze umístit i bez svých souřadnic na základě vlastnosti `grid-area`, která přiřadí item do pojmenované oblasti, která již byla explicitně specifikována vlastností `grid-template-areas`. Anebo pokud grid itemu nebyla specifikována oblast umístění, umístí se automaticky do prvního volného místa v mřížce dle vlastnosti `grid-auto-flow`, která již byla popsána v kapitole 3.2.2.

Na grid item můžeme aplikovat také vlastnosti pro zarovnání. První z nich je `align-self`, kterou lze použít na konkrétní item pro zarovnání ve sloupci, a je ekvivalentní k vlastnosti `align-items` grid kontejneru. Druhá z nich je `justify-self`, která také nic nového nepřináší, pouze zarovná konkrétní item v řádku, a je ekvivalentní k vlastnosti `justify-items` grid kontejneru.

3.4 Jednotka 'fr'

Grid Layout s sebou přinesl i novou jednotku, která nese zkratku fr. Tato zkratka nejspíše vychází z anglického slova fraction (zlomek), protože reprezentuje zlomek zbývajícího místa v grid kontejneru. Dá se spočítat jako poměr jedné části k poměru součtu všech částí. A využívá se pro definování velikosti řádků či sloupců mřížky. Pokud by někdy nastala otázka zda se tato jednotka dá aplikovat i u jiných typů layoutů, tak nejblíže má k Flexible Box Layoutu. Ve flexboxu se dá vytvořit podobný efekt této jednotky pomocí vlastností `flex-grow` a `flex-shrink`.

Za zmínku také stojí porovnání jednotky fr s vlastností `auto`. V občasných případech se jejich hodnoty můžou rovnat, ale jakmile se změní dostupné místo v grid kontejneru, tak se každá hodnota přepočítá, a jejich velikosti se už můžou lišit v řádech desítek pixelů. Z toho plyne, že fr a auto nemají stejné vlastnosti, protože fr si pro sebe zabere veškeré dostupné místo, které grid kontejner nabízí. Zatímco auto si zabere jen takové místo, aby dokázalo zobrazit celý svůj vnitřní obsah. [10]

Velikost jednotky fr se dá spočítat podle vzorce 3.1,

$$hypotheticalFR = \frac{DP - sumBS}{sumFF} \tag{3.1}$$

kde:

- hypotheticalFR – hypotetická (vypočtená) velikost jednotky fr
- DP – veškerý dostupný prostor, kde se může jednotka fr roztáhnout
- sumBS – součet všech řádků či sloupců, které mají pevnou velikost
- sumFF – součet všech jednotek fr v řádcích či sloupcích

3.5 Grid Layout vs Flexible Box

Důležitým poznatkem k zapamatování je, že Grid Layout, i když je novější, nenahrazuje Flexible Box Layout. Grid Layout se spíše využívá pro rozložení celé stránky oproti Flexible Boxu, který se spíše hodí na nějaké menší úseky stránky, ale není to pravidlem. Dalším rozdílem mezi těmito responzivními layouty je jejich účel dimenzování, Grid Layout je velmi silný pro dvojrozměrné rozložení, za to Flexible Box svůj účel plní pro jednorozměrné rozložení. Ve výsledku jsou tyto typy responzivního rozložení velice mocné pro tvorbu moderních webů a při vytváření webových stránek se doporučuje jejich kombinace. Bohužel Grid Layout oproti FLexible Boxu má stále ve webových prohlížečích menší podporu. [9]

Kapitola 4

Knihovna CSSBox

CSSBox je (X)HTML/CSS zobrazovací stroj napsaný v čisté Javě. Vznikl jako projekt na Fakultě informačních technologií Vysokého učení technického v Brně. Jeho primárním účelem je poskytnout kompletní a další zpracovatelné informace o vykresleném obsahu stránky a rozložení. Nicméně, může se však použít i pro prohlížení vykreslených dokumentů v Java Swing aplikací. Vstupem zobrazovacího stroje je DOM strom dokumentu a sada vlastností CSS, které jsou připojeny v hlavičce daného HTML dokumentu. Výstupem zobrazovacího stroje je objektově orientovaný model rozložení webové stránky. Tento model může být přímo zobrazen, ale primárně je vhodný pro další zpracování pomocí algoritmů rozložení analýzy, jako jsou například segmentace stránek nebo algoritmy extrakce informací. Jádro knihovny CSSBox může být také použito pro získání bitmapového nebo vektorového (SVG) obrazu vykresleného dokumentu. Díky balíčku SwingBox[1], může být CSSBox použit jako interaktivní webový prohlížeč v Java Swing aplikaci. CSSBox je licencován za podmínek - GNU Lesser General Public Licence version 3. [2]

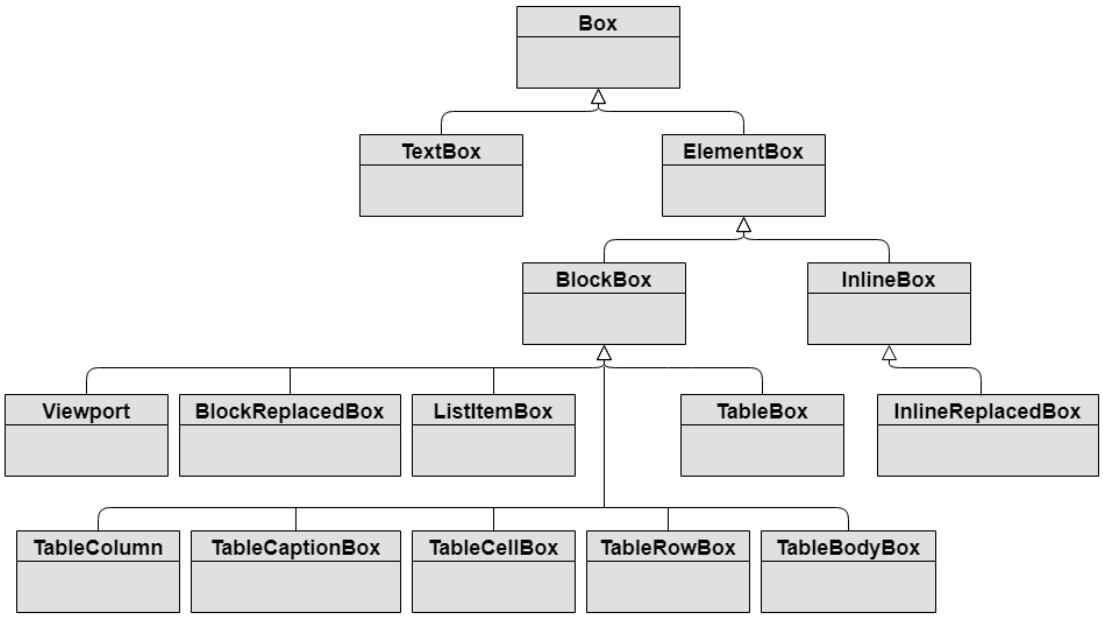
Všechny zdrojové kódy jsou dostupné ze serveru *github.com*[2]. Jelikož je celá knihovna napsaná pouze v jazyce Java, je díky tomu zaručena kompatibilita s velkou řadou operačních systémů, u kterých je jazyk Java podporován.

4.1 Model zobrazovaného dokumentu

Výsledné rozložení dokumentu je reprezentováno jako strom boxů. Každý box vytváří obdélníkovou oblast ve výsledné stránce a odpovídá tak konkrétnímu vykreslenému elementu jazyka HTML. Box může odpovídat nějaké určité části obsahu dokumentu, kterým může být například textový řetězec, nebo může být složen ze svých potomků. Každý box je reprezentován objektem, který rozšiřuje abstraktní třídu `Box`. V knihovně CSSBox aktuálně existuje několik typů boxů, které přibližně odpovídají vlastnosti CSS `display` pro daný element. Na obrázku 4.1 je zobrazena aktuální hierarchie boxů, se kterými knihovna pracuje a šipky v tomto obrázku znázorňují dědičnost mezi boxy.

Strom boxů má kořenový uzel, který je vždy reprezentován pomocí objektu `Viewport`, jenž představuje výřez prohlížeče. Ten má vždy jednoho potomka, který se nazývá kořenový box. Kořenový box odpovídá kořenovému elementu jazyka HTML, který je předán objektu `BrowserCanvas` na vykreslování, přičemž se obvykle jedná o element `<body>`.

[1] http://cssbox.sourceforge.net/swingbox/index.php
[2] https://github.com/radkovo/CSSBox



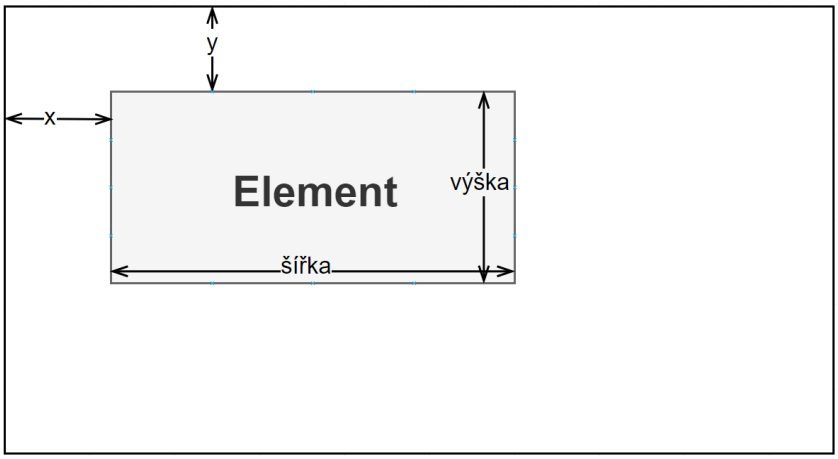
Obrázek 4.1: Ukázka aktuální hierarchie boxů v knihovně CSSBox. Obrázek byl převzat ze stránky[3] s manuálem a následně upraven pro lepší čitelnost.

4.2 Pozicování elementů v knihovně CSSBox

Na obrázku 4.2 je znázorněno pozicování elementů, které byly získány jako výstupní data knihovny CSSBox. Element je pozicován na základě svých souřadnic x a y, které se odvíjejí od levého horního rohu stránky. Souřadnice x udává vzdálenost elementu od levého okraje stránky a souřadnice y udává vzdálenost elementu od horního okraje stránky.

Na obrázku jsou dále znázorněny dva další důležité parametry elementů, kterými jsou výška a šířka elementu. Tyto parametry jsou důležité pro vykreslení elementu, pokud by nebyly známy, element by se nevykreslil (respektive vykreslil s nulovou velikostí) a umístil by se pouze na svojí pozici.

[3] http://cssbox.sourceforge.net/manual/



Obrázek 4.2: Pozicování elementů v knihovně CSSBox.

4.3 Odhalené nedostatky knihovny CSSBox

Při bližším studování architektury knihovny CSSBox bylo odhaleno také několik nedostatků, které mohou být překážkou pro správný vzhled výsledného dokumentu. Mezi tyto důležité nedostatky patří nepodporování desetinného formátu u jednotek, protože desetinná část je odříznuta (převedena na celočíselný typ). Dalším často vyskytujícím problémem jsou rozdílné rozměry textu v jednotlivých elementech. Nejsou podporovány například vlastnosti CSS `linear-gradient` či `border-radius`. Tlačítka rovněž nejsou podporována, pokud mají zobrazovat nebo skrývat nějaký text pomocí JavaScriptu, tak je text vykreslen mimo svůj přidělený prostor. V poslední řadě bylo zjištěno nepodporování protokolu HTTPS nebo práce s cookies.

4.4 Knihovna jStyleParser

jStyleParser [3] je knihovna napsaná v jazyce Java a je součástí projektu CSSBox. Funguje na bázi parseru s vlastním aplikačním rozhraní a využívá se pro analýzu stylů CSS. Tyto styly přiřazuje jednotlivým dokumentovým elementům jazyka HTML nebo XML podle specifikace CSS 2.1 a větší podmnožiny specifikace CSS 3. To umožňuje analyzovat jednotlivé soubory CSS, stejně jako výpočet efektivního stylu DOM elementů.

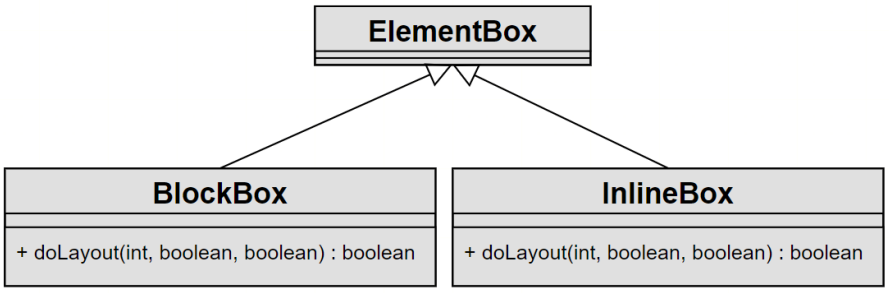
Kapitola 5

Návrh rozšíření knihovny CSSBox

Tato kapitola se nejprve věnuje popisu aktuální struktury dvou typů boxů. Následně se zaměřuje na úpravu této struktury boxů a zavedení nového systému Layout managerů. Poslední část je věnována návrhu rozšíření o podporu nových Grid Layout prvků.

5.1 Aktuální struktura třídy BlockBox a InlineBox

V aktuální struktuře těchto tříd je celkový způsob rozložení prováděn pomocí jejich metody `doLayout()`, kterou přímo implementují uvnitř sebe. Mezi hlavní nevýhody této struktury lze tedy považovat nemožnost přidělit jiný způsob jejich rozložení. Na obrázku 5.1 je zachycen diagram tříd znázorňující aktuální strukturu těchto tříd, včetně jejich metody `doLayout()`.



Obrázek 5.1: Diagram tříd znázorňující aktuální strukturu třídy `BlockBox` a `InlineBox`, včetně jejich způsobu rozložení.

5.2 Systém Layout managerů

Po domluvě s vedoucím byl předložen první požadavek, který požadoval upravit aktuální strukturu rozložení, tedy metodu `doLayout()`, tím způsobem, že tato struktura byla potřeba oddělit od dvou hlavních typů boxů, jimiž jsou `BlockBox` a `InlineBox`. Byl tedy navržen systém Layout managerů, který se kompletně stará o strukturu rozložení těchto boxů, a navíc se zde využívá tzv. zapouzdření, což se dá ve zkratce shrnout jako zaba-

lení dat a metod do jedné součásti. Zároveň tato celková úprava vedla k lepší přehlednosti a jednoduchosti zdrojového kódu.

Jako první bylo vytvořeno obecné rozhraní `org.fit.css.layout.LayoutManager`, které obsahuje hlavičku metody `doLayout()` a implementují ho třídy `org.fit.css.layout.BlockBoxLayoutManager` a `org.fit.css.layout.InlineBoxLayoutManager`. Ve výpisu 5.1 je ukázána struktura tohoto obecného rozhraní.

```
public interface LayoutManager() {
    boolean doLayout(int availw, boolean force, boolean linestart);
}
```

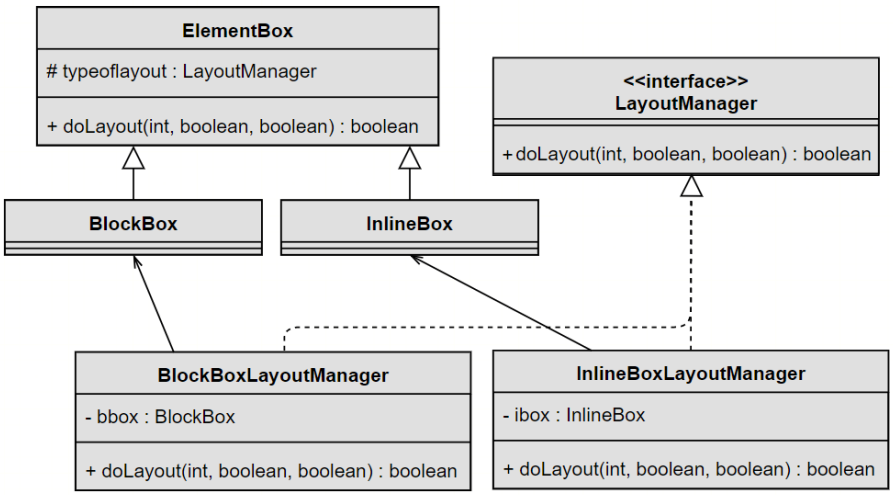
Výpis 5.1: Rozhraní `LayoutManager`.

Ve třídě `BlockBoxLayoutManager` je tedy obsažena metoda `doLayout()` a navíc je zde přidán atribut, který se stará o identifikaci daného BlockBoxu, včetně jeho konstruktoru. To samé platí pro třídu `InlineBoxLayoutManager`, kde je taktéž obsažena metoda `doLayout()` a atribut pro identifikaci InlineBoxu.

Dále byl přidán atribut s názvem `typeoflayout`, jenž je typu `LayoutManager`. Tento atribut je umístěn ve třídě `ElementBox`, aby k němu mohli přistupovat potomci, kterými jsou právě třídy `BlockBox` a `InlineBox`. Ty ho využívají pro přiřazení správného Layout manageru. Toto přiřazení se provádí již v konstruktoru daného boxu. Na obrázku 5.2 je zobrazen diagram tříd, který znázorňuje novou strukturu rozložení po přidání Layout managerů.

Tento systém Layout managerů je určitě vhodný přínos do budoucna. Už teď se může využít pro rozšíření implementace o podporu Flexible Box Layoutu. Nebo pokud by v budoucnu vyšel nový standard CSS pro rozložení, lze toto rozložení jednoduše začlenit do tohoto systému vytvořením nového Layout manageru.

Celkový systém Layout managerů byl navrhnut ve spolupráci se studentem Lukášem Ondrákem, který se rovněž zabývá rozšířením knihovny CSSBox [11]. Taktéž byla společně provedena následná implementace tohoto systému.



Obrázek 5.2: Diagram tříd znázorňující následnou strukturu třídy `BlockBox` a `InlineBox`, včetně jejich způsobu rozložení po přidání navrženého systému Layout managerů.

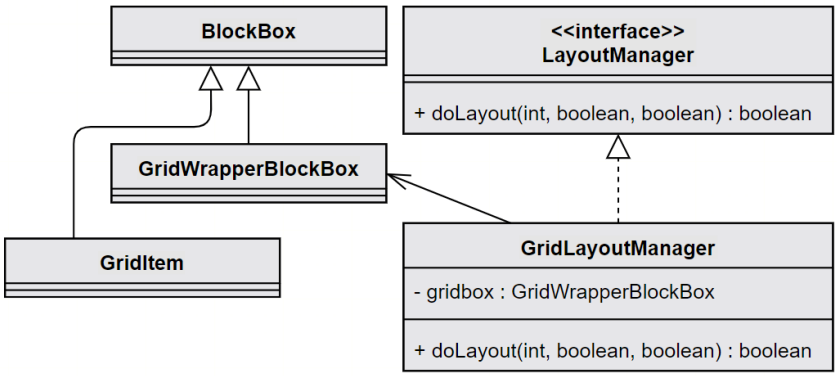
5.3 Začlenění Grid Layoutu do knihovny CSSBox

Pro správné fungování Grid Layoutu v knihovně CSSBox bylo nejprve potřeba navrhnout jeho příslušný Layout manager a začlenit ho do systému managerů. Byla tedy navrhnuta třída `org.fit.css.layout.GridLayoutManager`, která taktéž bude implementovat obecné rozhraní `LayoutManager`. Následně v něm bude implementována metoda `doLayout()` podle algoritmu Grid Layout. Zároveň do této třídy bylo potřeba začlenit i atribut, včetně jeho konstruktoru, který reprezentuje příslušný grid kontejner.

Dále bylo potřeba navrhnout příslušnou strukturu pro dvě hlavní složky Grid Layoutu, kterými jsou kontejner a item. Byla vytvořena třída `org.fit.css.layout.GridWrapperBlockBox`, která reprezentuje kontejner a rozšiřuje třídu `BlockBox`. Byl vytvořen i konstruktor, ve kterém je rovněž přidán atribut `typeoflayout` pro správné přiřazení Layout manageru. Pro item byla vytvořena třída `org.fit.css.layout.GridItem`, včetně jejího konstruktoru, která taktéž rozšiřuje třídu `BlockBox`. Na obrázku 5.3 je zobrazen diagram tříd znázorňující novou strukturu tříd pro Grid Layout. Tato struktura, tvořená právě grid kontejnerem a itemem, zároveň rozšiřuje i aktuální hierarchii boxů v knihovně CSSBox, která již byla vyobrazena na obrázku 4.1.

V poslední řadě byla navrhnuta a následně i vytvořena pomocná třída `org.fit.css.layout.GridItemRowColumn`, která bude sloužit pro uchovávání souřadnic jednotlivých itemů.

Následná práce s těmito novými třídami bude podrobněji popsána v následující kapitole.



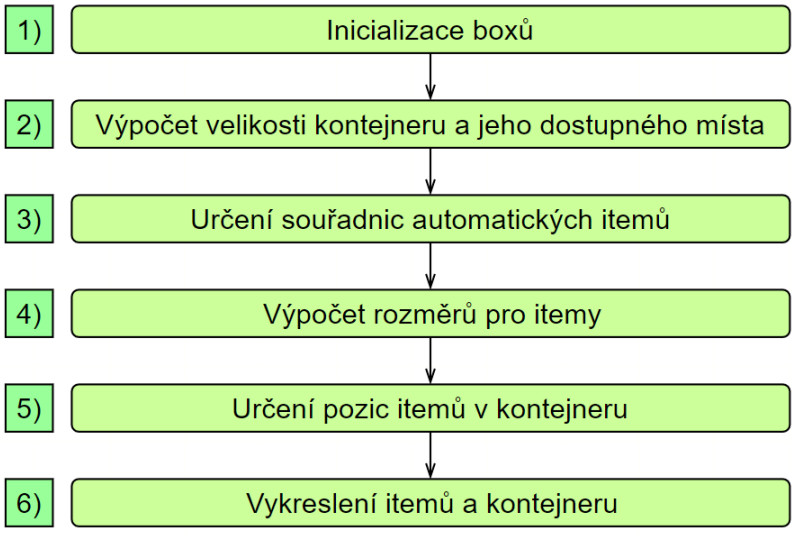
Obrázek 5.3: Diagram tříd znázorňující začlenění struktury Grid Layout do nového systému Layout managerů.

5.4 Návrh algoritmu Grid Layout

Tato podkapitola se zabývá návrhem algoritmu Grid Layout. Jako inspirace pro tento návrh byly využity oficiální stránky konsorcia W3C[1], kde je popsán obecný algoritmus Grid Layout. Na obrázku 5.4 jsou znázorněny jednotlivé kroky algoritmu a následně jsou rozepsány pro lepší přiblížení.

Grid Layout algoritmus lze rozdělit do šesti částí. V následujících odstavcích jsou tyto části podrobněji popsány.

[1] https://www.w3.org/TR/css-grid-1/#algo-overview



Obrázek 5.4: Diagram znázorňující jednotlivé kroky algoritmu Grid Layout.

První čast algoritmu se zaměřuje na inicializaci příslušných boxů, včetně potřebných anonymních boxů. Pro kontejner jsou načítány všechny jeho náležité styly CSS, které jsou ve formě Java objektů. V případě itemů je proces načítání stylů CSS prováděn obdobně. Zároveň se tato část zabývá i zjišťováním souřadnic pevně umístěných itemů v kontejneru a získáváním velikosti stop explicitní a implicitní mřížky.

Druhá část algoritmu se zabývá velikostí kontejneru, která je potřebná pro následné operace. Dále se tato část zabývá dostupným prostorem v kontejneru, který je potřeba zjistit pro další část algoritmu.

Třetí část algoritmu řeší problematiku automatických itemů. Jelikož ještě neznají své souřadnice pro umístění, tak jim budou přiděleny na základě volného prostoru v kontejneru.

Čtvrtá část algoritmu se stará o správný výpočet rozměrů itemů, tedy jejich šířku a výšku. V této části už jsou známy všechny potřebné údaje pro tento výpočet, kterými jsou souřadnice itemu, velikosti stop kontejneru a v poslední řadě i mezery mezi stopami.

Pátá část algoritmu počítá vzdálenosti itemů od levého horního rohu kontejneru. Pro tento výpočet jsou rovněž potřeba znát údaje o velikosti stop kontejneru, mezer mezi nimi a startovních souřadnic itemů.

Šestá část algoritmu ukládá vypočítané hodnoty ze čtvrté a páté části do příslušných struktur, které následně itemy vykreslí. Je zde ještě potřeba zkontrolovat finální velikosti kontejneru, než bude vykreslen, protože může být i explicitně ovlivněn pomocí vlastností CSS, kterými jsou například `width`, `height` nebo jim podobné.

Kapitola 6

Implementace

V této kapitole jsou nejprve probrány nezbytné úpravy v knihovně CSSBox pro podporu potřebných boxů. Následuje detailní popis implementace algoritmu Grid Layout z kapitoly 5.4. Závěr této kapitoly je věnován nedostatkům v implementaci.

6.1 Úprava třídy ElementBox a BoxFactory

Dle návrhu systému Layout managerů z předchozí kapitoly a pro správné chování managerů bylo ještě potřeba přidat metodu `doLayout()` do třídy `ElementBox`, protože tato třída je nadřazená všem třídám, kterých se nový systém Layout managerů týká. Tato metoda akorát přiřazuje příslušný typ manageru a následně volá metodu pro rozložení dle určitého typu boxu. Ve výpisu 6.1 je ukázka této metody a způsob její konstrukce.

```
@Override
public boolean doLayout(int availw, boolean force, boolean linestart) {
     LayoutManager lm = typeoflayout;
     lm.doLayout(availw, force, linestart);
     return true;
}
```

Výpis 6.1: Metoda `doLayout()` ve třídě `ElementBox`.

Pro správné generování potřebných typů boxů byla potřeba upravit i třída `BoxFactory`. V metodě `createElementInstance()`, která vytváří instance třídy `ElementBox` na základě jejich hodnoty pro vlastnost display, se nevyskytuje podpora hodnoty grid. Tudíž bylo zvoleno rozšíření, které mezi rozhodovací podmínky vložilo podporu vlastnosti `display:grid`. Díky tomuto zásahu se již dokáže vytvářet potřebný grid kontejner. Navíc je v této metodě ještě řešena problematika grid itemů, které se vytvoří pouze pokud jsou přímí potomci grid kontejneru.

Důležitým prvkem v této třídě je také zaručení správného vytváření anonymních boxů pro grid itemy, které již byly zmíněny v kapitole 3.3. Byl tedy nutný zásah do metody `createAnonymousBox()`, která se stará o vytváření příslušných anonymních boxů. Tato metoda byla rozšířena o vytváření anonymních boxů pro grid itemy, pokud je rodičovským elementem grid kontejner.

6.2 Získání stylů CSS pro grid kontejner a item

Tato podkapitola se nejprve zabývá načítáním stylů CSS pro grid kontejner. Na tuto část navazuje určování velikostí stop, kde je i zmíněna implementace jednotek fr. V poslední části je probíráno načítání stylů CSS pro grid item.

Grid kontejner

Pro načítání stylů CSS bylo využito metody `setStyle()`, která zprostředkovává načítání stylů pro konkrétní box. Zde načte všechny styly, včetně stylů jeho předků, které se týkají daného kontejneru. Zpracování jednotlivých stylů pro kontejner je rozděleno do metod kvůli přehlednosti zdrojového kódu. K získání skutečné hodnoty stylu CSS je využito zápisu `style.getProperty("nazev-stylu-css")`, který poskytuje knihovna jStyleParser. Styly samozřejmě nemusí být vždy zadány, tudíž bylo zvoleno řešení, kdy se příslušnému stylu přiřadí jeho výchozí hodnota podle specifikace CSS. Po získání hodnoty se dále rozhoduje, jak se kontejner bude chovat ve výsledku vykreslování. Od toho se také odvíjí způsob zpracování hodnoty daného stylu.

Například dle specifikace tvoří mezery mezi stopami číselné hodnoty. Byl tedy zvolen způsob zpracování hodnoty pomocí třídy `CSSDecoder`[1] a metody `getLength()`, která si načte hodnoty a převede je do pixelových jednotek. Tato hodnota je pak následně přiřazena k náležitému atributu v kontejneru.

U složitějších stylů jsou pak navíc využívány tzv. `TermListy`, které v sobě dokáží udržet různé kombinace hodnot, včetně názvů jednotek u číselných hodnot. Tyto `TermListy` poskytuje knihovna jStyleParser a zároveň slouží jako pomocná struktura při určování velikostí stop explicitní mřížky.

Velikost stop explicitní a implicitní mřížky

Celá velikost explicitní mřížky (řádkové i sloupcové stopy) je uložena v již zmíněné struktuře `TermList`. Od tohoto kroku tedy lze zjistit velikosti jednotlivých stop mřížky. Každá stopa je uložena na pozicích, ke kterým lze přistupovat pomocí indexové hodnoty, počínajíc od nuly. Jakmile je znám index pro požadovanou stopu, tak na této pozici je potřebná hodnota taktéž zpracována metodou `getLength()`. Toto je prováděno pouze, pokud všechny stopy jsou definovány pomocí jednotek.

V opačném případě, kdy stopy obsahují jednu z hodnot `auto`, `min-content`, `max-content` nebo jejich kombinaci, je nejprve potřeba znát obsah každého itemu. Tento proces je dále popsán v kapitole 6.5.

Implicitní mřížku je zbytečné ukládat do struktury `TermList`, protože se skládá pouze z jedné hodnoty, jak pro stopu řádku tak i sloupce. Jestliže se jedná o číselnou stopu mřížky, její hodnota je jednoduše získána skrze metodu `getLength()` a následně uložena do příslušného atributu kontejneru. V ostatních případech jsou detekce hodnoty prováděny pomocí atributů, které jsou typu *boolean*.

Pokud se v explicitní mřížce vyskytne funkce `repeat()`, je následně zpracována a uložena to tzv. `RepeatImpl` listu. Díky tomuto listu lze jednoduše zjistit počet opakování pomocí metody `getNumberOfRepetitions()` a jaké velikosti se mají opakovat obstarává metoda `getRepeatedTerms()`.

[1] `CSSDecoder - tato třída implementuje konverzi specifikací CSS do datových typů jazyka Java`

Implementace jednotek fr

Před samotnou implementaci jednotek fr byla nejprve přidána jejich podpora ve třídě `VisualContext`, přesněji v metodě `pxLength()`. Zde byla rozšířena konstrukce *switch*, kde byl přidán nový *case*, který vrací aktuální hodnotu jednotky fr.

Při implementaci jednotek fr bylo vycházeno ze vzorce 3.1. Jako první krok v implementaci bylo zvoleno nalezení všech jednotek fr, které se vyskytují ve stopách mřížky , poté byl udělán jejich součet. Dalším důležitým krokem bylo sečtení všech stop, které mohou bránit ve vyplnění volného prostoru. Nakonec od celkové velikosti kontejneru je nutné odečíst sumu spočítaných hodnot stop včetně vlastností margin, border a padding, tím je zjištěn volný prostor pro usazení obsahu.

Díky těmto operacím již je možné spočítat výslednou hodnotu 1fr, kde se jen vydělí volný prostor součtem všech jednotek fr. Tato hodnota je uložena do atributu kontejneru, protože v následujících krocích bude využívána pro určování velikosti itemů či bude hrát roli v jejich umístění.

Grid item

Pro následující zpracovávání souřadnic jednotlivých itemů nejprve byla implementována třída `GridItemRowColumn`. Tato třída dle návrhu bude v sobě uchovávat veškeré souřadnice itemů. Souřadnice jsou obecně tvořeny číselnými hodnotami, díky tomuto tvrzení byly vytvořeny čtyři atributy typu *int*. Itemy ale nemají vždy pevné souřadnice, mohou se i libovolně roztahovat. Na zpracování tohoto roztahování byly taktéž vytvořeny čtyři atributy, které jsou typu `TermList`.

Zde rovněž bylo využito metody `setStyle()` pro načtení všech stylů CSS grid itemů. Stejně jako u grid kontejneru k získání hodnoty stylu je použit zápis `style.getProperty("nazev-stylu-css")`. Nezadaný styl je také zpracován přiřazením výchozí hodnoty dle specifikace CSS. Aby byly podporovány tyto hodnoty, byla ještě rozšířena konstrukce *switch* v metodě `pxLength()` o nový *case*, který vrací tyto hodnoty jako bez jednotkové.

Zpracování souřadnic bude vysvětleno na souřadnicích pro sloupce, následně tento proces je obdobný i na souřadnice pro řádek. Číselné souřadnice jsou tedy zjištěny a jednoduše zpracovány pomocí metody `getLength()` a následně uloženy do jednotlivých atributů třídy `GridItemRowColumn`. Bohužel zde nastává i několik problémů. Koncová souřadnice nemusí být zadána, je nutné ji tedy dopočítat, výpočet je proveden přičtením čísla jedna vůči své startovní souřadnici. Stejně tak nemusí být zadána startovní souřadnice, v tomto případě se jen odečte číslo jedna od koncové souřadnice. Dále může nastat situace kdy koncová souřadnice je vyšší než startovní. Je tedy nutné tyto dvě hodnoty mezi sebou prohodit, například pomocí tzv. swapování hodnot přes třetí proměnnou, které bylo rovněž zvoleno v implementaci.

Další část implementace se zabývá souřadnicemi na základě roztahování, jedná se tedy o hodnotu *span*. Celá hodnota je uložena ve span atributu. Aby se zjistila její číselná hodnota, je ji potřeba vyhledat ve struktuře `TermList` na pozici jedna. V případě koncové souřadnice je tato hodnota přičtena ke startovní souřadnici. V konkrétní situaci by to vypadalo následovně: startovní souřadnice obsahuje hodnotu dva, koncová souřadnice zase hodnotu span tři, výsledek koncové souřadnice tedy bude pět. V opačném případě je hodnota span odečtena od koncové souřadnice a přiřazena do startovní.

V poslední řadě se mohou objevit itemy, které nemají zadané žádné souřadnice nebo obsahují hodnotu auto. Tyto itemy jsou považovány za automaticky umístěné itemy, do jejich atributů pro souřadnice je uložena nulová hodnota a jsou dále řešeny v kapitole 6.4.

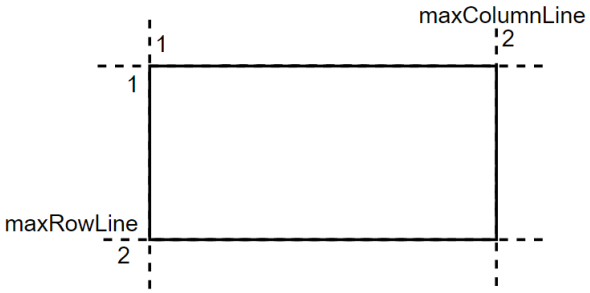
6.3 Velikost grid kontejneru a jeho dostupný prostor

Základní velikost kontejneru je nastavena na velikost 1x1, respektive pro účely implementace a umisťování itemů 2x2, tyto hodnoty představují atributy `maxColumnLine` a `maxRowLine`, které zároveň reprezentují linky mřížky zmíněné v kapitole 3.1. Na obrázku 6.1 je takový kontejner vyobrazen, včetně jeho pomocných atributů.

Pro zjištění aktuální velikosti kontejneru je potřeba nejprve prohledat všechny jeho grid itemy vyskytující se v něm. Při prohledávání itemů je nezbytné si ukládat koncové souřadnice pevně umístěných itemů do příslušných atributů, aby se mezi nimi vybraly největší souřadnice, které zároveň budou reprezentovat nejvyšší rozměr kontejneru. Automaticky umístěné itemy jsou v tomto kroku ignorovány.

Pokud by nastal případ, kdy je vytvořena v kontejneru kombinace explicitní a implicitní mřížky při pozicování automatických itemů, je potřeba zjistit ještě směr umisťování itemů, od kterého se bude odvíjet výsledný rozměr mřížky. Jestliže bude směr umisťování pro řádek, tak do atributu `maxColumnLine` je uložena hodnota, kterou tvoří počet sloupcových stop explicitní mřížky zvětšen o jedna. Při umisťování pro sloupec je tento proces obdobný.

Tvoří-li kontejner pouze automaticky umístěné itemy, tak dle směru umisťování itemů je do příslušného atributu nahrána základní hodnota kontejneru. A do druhého atributu se nahraje hodnota, která představuje počet itemů zvětšen o jedna. Ve výsledku v tomto případě se kontejner jeví jako jeden řádek či sloupec.



Obrázek 6.1: Ukázka upraveného grid kontejneru, který vychází z upravené základní mřížky. Upravená mřížka se nevztahuje nikoliv na svoje řádky a sloupce, které tvoří dohromady rozměr 1x1, ale její rozměry jsou určeny až od koncových linek mřížky, tedy velikost 2x2.

Dostupný prostor

K určení dostupného prostoru v kontejneru bylo využito tzv. Collections[2], které jsou běžnou součástí jazyka Java, přesněji byl zvolen seznam, protože umožňuje přístup k libovolnému prvku a podporuje tři důležité metody, jimiž jsou `add()`, `remove()` a `get()`. Dostupný prostor kontejneru je potřeba znát pro automaticky umístěné itemy.

Prvním krokem je potřeba naplnit seznam hodnotami začínající od čísla jedna po maximální velikost kontejneru zmenšenou o jedna. Tyto hodnoty budou představovat budoucí startovní souřadnice pro automaticky umístěné itemy a zároveň se dají považovat za pomocné linky mřížky, které mají stejné vlastnosti jako linky popsané v kapitole 3.1.

[2]Collections – poskytují strukturu pro ukládání a manipulaci se skupinou objektů, využívají základních operací nad daty (vyhledávání, vkládání, mazání, třídění)

Následně je nutné vyjmout ze seznamu všechny souřadnice, které jsou již obsazené. Nejprve jsou odstraněny takové hodnoty, které patří startovním souřadnicím pevně umístěných itemů rozkládajících se do nejvýše jedné oblasti mřížky.

Následuje odstranění dalších hodnot, jenž se vztahují k souřadnicím pevně umístěných itemů ve dvou a nebo více oblastí mřížky. U těchto itemů je zároveň důležité provádět kontrolu jejich souřadnic v předešlých řádcích či sloupcích, aby nedošlo k odstranění špatné hodnoty v aktuálním řádku či sloupci.

Nyní je znám veškerý dostupný prostor kontejneru pro následující úkony. Dá se automaticky rozšiřovat podle potřeby, čímž zároveň rozšiřuje i celkovou velikost kontejneru.

6.4 Souřadnice automaticky umístěných položek

Určování souřadnic automaticky umístěných itemů je prováděno na základě dostupného místa v kontejneru. Nastane-li situace, kdy je velký počet itemů a dostupný prostor je již nedostatečný, dojde k rozšíření dostupného prostoru příslušným způsobem, tedy kontejner je rozšířen o nové řádky či sloupce.

Automaticky umístěné itemy mohou zabírat pouze jednu oblast mřížky, od tohoto tvrzení se také odvíjí příslušné přidělení souřadnic. Při tomto procesu je rovněž využita pomocná třída `GridItemRowColumn`, která bude uchovávat nově přidělené souřadnice.

Je-li zvolen směr umisťování itemů do řádku, první přidělenou hodnotou do startovní souřadnice pro řádek je číslo aktuálního volného řádku. Následně je nutné dopočítat koncovou souřadnici pro řádek. Výpočet je proveden zvětšením o jedna vůči své startovní souřadnici.

Nyní je potřeba dopočítat sloupcové souřadnice. Jelikož jsou již známé řádkové souřadnice, předpokládá se, že v aktuálním řádku je volný prostor. Startovní souřadnice pro sloupec je k nalezení na nulté pozici v již zmíněném seznamu. Pro získání této hodnoty je využito metody `get()`, pomocí které je tato hodnota získána a přiřazena do startovní souřadnice. Následně je potřeba tuto hodnotu smazat ze seznamu, aby se další itemy na tento prostor dále nepřiřazovali. Koncová souřadnice pro sloupec je rovněž zvětšena o jedna oproti své startovní.

Tento proces přiřazování je stejný i v případě směru umisťování itemů do sloupce s rozdílem, že v seznamu jsou uloženy řádkové souřadnice.

Aby vůbec byla detekována situace přeplnění kontejneru, je vždy prováděna kontrola, zda zbývá nějaký automaticky umístěný item, který ještě nemá přiděleny svoje souřadnice. Pokud takový existuje je pro něj a všechny jeho následující itemy rozšířen dostupný prostor o nový řádek či sloupec dle směru umisťování itemů a celý proces přiřazování souřadnic se opakuje pro zbývající itemy.

6.5 Výpočet rozměrů položek a jejich umístění v kontejneru

V této podkapitole je nejprve popsán výpočet rozměrů jednotlivých itemů z číselných stop mřížky. Poté se řešen problém kdy stopy neobsahují jen číselné velikosti. Konec této podkapitoly je věnován principu umístění jednotlivých itemů v kontejneru.

Výpočet šířky a výšky

V tomto kroku jsou již známy všechny souřadnice itemů, lze se tedy pustit do určování šířek a výšek (viz. obrázek 4.2) jednotlivých itemů na základě velikostí stop mřížky.

Výpočet šířky i výšky je prováděn pomocí cyklu, který začíná na startovní souřadnici itemu a končí na koncové souřadnici zmenšeno o jedna. V každém kroku cyklu je nejprve řešena situace v jaké mřížce se aktuálně řešená souřadnice nachází a následně je probírána její velikost stopy. V případě implicitní mřížky je pouze přičítána její aktuální velikost stopy, dle právě zpracovávané souřadnice do celkového rozměru itemu. V případě explicitní mřížky je postupováno stejným způsobem. Zajímavá situace nastává, když je zadána kombinace těchto mřížek. Nejdříve je třeba zjistit, jak velká je explicitní mřížka. Ta je uložena v již zmíněném `TermList`u a jeho velikost je zjištěna pomocí metody `size()`, tato hodnota bude udávat hranici mezi těmito mřížkami. Ke všem souřadnicím, které nabývají menší hodnotu než tato hranice, jsou hledány příslušné velikosti stop v explicitní mřížce pomocí metody `get()`, kde jejím parametrem je číslo aktuální souřadnice. Nalezená velikost stopy je nakonec přičtena do celkového rozměru itemu. Všem souřadnicím, které nabývají vyšší nebo stejnou hodnotu než je hraniční hodnota, jsou nacházeny velikosti stop v implicitní mřížce a jejich hodnoty jsou přičítány do celkového rozměru itemu.

Pokud je item roztažen přes dvě a více stop, jsou do jeho rozměru započítány i mezery mezi stopami. Přes kolik mezer je item roztažen, je jednoduše zjištěno odečtením startovní souřadnice itemu od koncové a následným odečtením hodnoty jedna. Poté je výsledná hodnota vynásobena hodnotou, představující velikost mezery mezi stopami, a přičtena do celkového rozměru itemu.

Pokud nějaká stopa obsahuje jednotku, fr tak její hodnota je nejprve zjištěna a poté vynásobena s příslušným atributem kontejneru, reprezentující hodnotu 1fr převedenou do pixelových jednotek. Nakonec je přičtena do celkového rozměru itemu. Implementace jednotek fr již byla popsána v kapitole 6.2.

Následující odstavce se zabývají výpočty rozměrů itemů v případě hodnot `auto`, `min-content`, `max-content`, které jsou zadány v jednotlivých stopách.

Ve sloupcových stopách pro zjištění minimální či maximální šířky itemu bylo využito metod `getMinimalWidth()` a `getMaximalWidth()`, které jsou součástí třídy `BlockBox`. V případě hodnoty `auto` je dočasně nastavená šířka itemu pomocí nulové hodnoty. Nyní jsou známy všechny šířky itemů, které se odvíjejí od svého obsahu. Bohužel tyto šířky nemusí být vždy stejné, je tedy potřeba zaručit, aby všechny itemy ve sloupci měly stejné hodnoty.

Na vyřešení tohoto problému byla vytvořena metoda `checkColumnLine()`, která využívá parametru pro detekci určitého sloupce. Tato metoda nejprve kontroluje celý sloupec a hledá nejširší item, který je usazen pouze do jednoho sloupce. Šířku tohoto itemu prohlásí za výslednou pro celý sloupec a následně tuto hodnotu přidělí všem itemům v daném sloupci (rovněž se jedná o itemy usazené pouze do jednoho sloupce).

Hodnoty z každého sloupce jsou následně uloženy do pomocného seznamu, jenž nakonec bude reprezentovat všechny rozměry jednotlivých stop. Je zde uložena i nulová hodnota pro hodnotu auto, pokud existuje.

Při řešení rozměru hodnoty `auto` bylo vycházeno ze vzorce 6.1. Dostupný prostor zde zastupuje celková velikost obsahu kontejneru. Dále je potřeba spočítat z pomocného seznamu počet výskytů hodnoty `auto` (ta je stále reprezentována nulovou hodnotou) a sečíst všechny nenulové hodnoty. V poslední části jsou ještě dopočítány mezery mezi stopami, tak že je zjištěna velikost pomocného seznamu, od které je odečtena hodnota jedna, a následně vynásobena velikostí mezery.

Výsledná hodnota `auto` je uložena do pomocného seznamu na všechny pozice, které obsahují nulovou hodnotu. Tím jsou kompletně vyřešeny všechny velikosti sloupcových stop.

$$autoHodnota = \frac{dostupnyProstor - gridGap - soucetStopBezAuto}{pocetAutoStop} \tag{6.1}$$

V poslední řadě je potřeba vyřešit šířky zbývajících itemů, které jsou rozloženy přes dvě a více stop. To je provedeno rovněž v cyklu jak již bylo řečeno na začátku této podkapitoly, s tím rozdílem, že požadované hodnoty jsou na základě indexu hledány v pomocném seznamu.

V řádkových stopách je proces prováděn podobně. Je zde ale několik rozdílů. Hodnoty `auto` a `max-content` jsou dle specifikace počítány stejně jako hodnota `min-content`. Tudíž všechny tři hodnoty jsou zpracovány stejně pomocí metody `getContentHeight()`, kde jsou ještě připočítány vlastnosti, kterými jsou horní a dolní margin, padding a border. Je zde i kontrola pro pevné itemy, které mohou zasahovat až za celkovou mřížku. Kontrola je prováděna pomocí konstrukce `try` a `catch`, kdy jsou nejprve brány velikosti z pomocného seznamu, a pokud se narazí na souřadnici, která je mimo seznam, přičte se akorát velikost mezery mezi stopami řádků. V tomto seznamu jsou uloženy všechny hodnoty stop explicitní i implicitní mřížky.

Nakonec je dobré zmínit, že nemusí být zadána ani jedna z mřížek, v tomto případě je šířka itemů roztažena po celé šířce kontejneru a výška je odvíjena od velikosti obsahu itemů.

Umístění v kontejneru

V tomto bodě je řešena velikost souřadnic x a y (viz. obrázek 4.2), které využívá knihovna CSSBox pro umisťování všech boxů.

Výpočet vzdáleností itemů od levého horního rohu kontejneru, tedy od souřadnic [0, 0], je prováděn pomocí cyklu. Cyklus je nastaven na počáteční hodnotu nula a jeho konec reprezentuje startovní souřadnice daného itemu zmenšena o jedna. Zde je také nejprve řešena problematika mřížek. V případě pouze implicitní mřížky je do souřadnice x a y postupně přičítána velikost stopy až do konce cyklu. Tento samý postup je prováděn i v případě pouze explicitní mřížky. Zajímavější je kombinace těchto mřížek, také je zde potřeba nejprve zjistit velikosti explicitní mřížky, od které se bude odvíjet další zpracování.

Všechny souřadnice nabývající menší hodnoty, než je velikost explicitní mřížky, detekují zpracování vzdálenosti postupným přičítáním jednotlivých velikostí stop mřížky do výsledných souřadnic x a y. V opačném případě je nejprve spočítána celková velikost explicitní mřížky a poté uložena do souřadnic x a y. Do těchto souřadnic je ještě následně přičtena vzdálenost v implicitní mřížce.

Po určení jednotlivých vzdáleností je v závěru tohoto procesu ještě provedena implementace mezer mezi stopami, se kterými je nutno také počítat v celkové vzdálenosti. Počet mezer je jednoduše zjištěn pomocí startovní souřadnice itemu, od které je odečtena hodnota jedna. Tento počet je následně vynásoben hodnotou, která představuje velikost mezery, a nakonec přičten do souřadnic x a y.

Pokud byla zadána některá z hodnot `auto`, `min-content` nebo `max-content`, tak výpočet vzdálenosti od levého horního rohu kontejneru je prováděn rovněž v cyklu, jak již bylo zmíněno v úvodu. Jediný rozdíl spočívá v tom, že jednotlivé vzdálenosti jsou brány z pomocného seznamu, jenž byl zmíněn už při určování šířky a výšky, a nikoliv přímo z velikostí stop.

6.6 Vykreslení kontejneru a položek

Před samotným vykreslením kontejneru je ještě potřeba zkontrolovat jeho finální výšku. Jak již bylo řečeno jeho výška může být ovlivněna pomocí vlastností CSS `height`, `min-height` nebo `max-height`. Pokud je zadána vlastnost `height`, kontejner převezme tuto hodnotu a prohlásí ji za svou výšku. Při zadané vlastnosti `max-height` je prováděna kontrola zda aktuální velikost kontejneru nepřesahuje tuto hodnotu, pokud by náhodou byla přesažena, tak kontejneru je určena tato hodnota jako výška. V opačné situaci je považována za výšku jeho aktuální výška. V případě vlastnosti `min-height` je kontrolována rovněž jeho aktuální výška. Je-li větší než minimální výška, tak za jeho výšku je považována aktuální.

Výška kontejneru je ještě kontrolována vůči pomocnému seznamu z kapitoly 6.5, který může obsahovat i nevyužité stopy explicitní mřížky podílející se rovněž v určení výsledné výšky kontejneru.

Přiřazení výšky kontejneru je prováděno pomocí metody `setContentHeight()` a následně do atributu `bounds.height` je přičtena jeho výška včetně příslušných vlastností margin, padding a border. Tím je zajištěna finální výška kontejneru a kontejner je následně vykreslen.

Před finálním vykreslením itemů je nejprve potřeba uložit jejich rozměry a umístění v kontejneru do příslušných struktur, které jsou použity pro vykreslení. Do atributů `content.width` a `content.height` je uložena celková velikost itemu, po odečtení vlastností margin, padding a border. Dalšími důležitými atributy jsou `bounds.width` a `bounds.height`, do kterých je potřeba uložit celkovou velikost itemu. Posledním důležitým krokem je uložení zjištěných vzdáleností itemu od levého horního rohu. To je prováděno za pomocí metody `setPosition()`, kde prvním parametrem je již zmíněná souřadnice x a druhým parametrem souřadnice y.

Vykreslení obsahu itemů

Na vykreslení obsahové části jednotlivých itemů byly využity metody `layoutInline()` a `layoutBlock()`, které jsou již implementovány ve třídě `BlockBox`. Jaká metoda je zrovna použita rozhoduje atribut `contblock`, indikující zda daný item obsahuje nebo neobsahuje blokové elementy. Před samotným voláním těchto metod je zapotřebí nejprve uložit šířku obsahové části itemu do atributu `availwidth`, který reprezentuje dostupný prostor pro rozložení všech elementů uvnitř itemu, a to je provedeno za pomocí metody `setAvailableWidth()`.

6.7 Co nebylo implementováno

Práce se nejprve zabývala celkovou implementací Grid Layoutu pomocí pevných jednotek, včetně rozmisťování itemů. Poté byla rozšiřována o nové prvky velikostí stop jako jsou například hodnoty `min-content`, `max-content` a `auto`. Nebyla tedy implementována podpora umisťování itemů na základě pojmenovaných oblastí. Dále nebyla implementována funkce `minmax()`,rozšířená podpora funkce `repeat()` a vlastnost `order`. Tato vlastnost se dá poměrně nahradit pomocí vlastnosti `grid-auto-flow` nebo její kombinací s pevně umístěnými itemy.

Kapitola 7

Testování

Tato kapitola se věnuje testování implementovaného rozšíření knihovny CSSBox. Začátek kapitoly je zaměřen na testování pomocí testovací sady. Další část je věnována testování na základě vytvořených vlastních testovacích stránek. Závěr této kapitoly shrnuje dosažené výsledky a pojednává o dalších možnostech rozšíření.

7.1 Testovací sada

V rané fázi testování bylo nejprve řešeno, jak nejlépe a vhodně otestovat implementované rozšíření. Pro účely testování byla tedy pořízena vhodná testovací sada ze serveru *github.com*[1] od uživatele *jgraham*. Tato testovací sada je sice starší verze, ale poskytuje potřebné testy pro otestování algoritmu Grid Layout. Oficiální testovací sada je k nalezení na stránkách[2] organizace W3C, kde lze nalézt i další testovací sady pro různé moduly CSS.

V knihovně CSSBox se nacházejí pomocné nástroje, které umožňují spouštět tyto testovací sady. Přesněji se jedná o třídu `TestBatch`. Tato třída poskytuje snadnější testování, umožňuje spouštění více testů najednou a ve výsledku hodnotí úspěšnosti jednotlivých testů.

Pro účely testování bylo nakonec využito právě této třídy. Před samotným spuštěním bylo potřeba nejdříve provést několik úkonů. Nejprve byly všechny testy z testovací sady přidány do souboru `test_reference.csv`, který slouží pro uložení referenčních hodnot. K testům bylo ještě potřeba přidat maximální přípustnou odchylku. Odchylka byla zvolena minimální, tedy *0,0*. Musela být specifikována i cesta k souboru `reftest-toc.htm`, protože v něm jsou obsaženy všechny testy pro jejich následné načítání.

Při spuštění jsou nejprve načítány všechny referenční hodnoty včetně svých odchylek, které jsou následně použity pro porovnávání s aktuálně bežícími testy. Všechny testy, které jsou vyhodnoceny s přípustnou odchylkou jsou považovány za úspěšné. V případě neúspěšných testů je na standardní chybový výstup vypsaný název testu s odchylkou, reprezentující jak moc se výsledek liší od původu.

Při důkladné kontrole běžících testů byla ale objevena zásadní chyba ve třídě `TestBatch`. Testy, které nedoběhly do konce a někde v průběhu zpracování se vyskytla chyba, byly považovány za úspěšné. V tabulce 7.1 je zobrazena statistika výsledků automatického testování na testovací sadě. Na základě výsledné statistiky, která přinesla pouhých 56,862 %

[1] https://github.com/jgraham/css-test-built/tree/master/css-grid-1_dev
[2] http://test.csswg.org/harness/

úspěšnosti, byla provedena následná ruční kontrola všech dosavadních testů, tato kontrola pomohla ke zlepšení výsledné statistiky.

Po objevení zmíněné chyby bylo prováděno další testování za pomocí testovací sady již ve webovém prohlížeči Google Chrome verze 73.0. Zde byl zobrazen referenční test a následně ten samý byl puštěn v knihovně CSSBox. Pro zjištění jednotlivých velikostí boxů ve webovém prohlížeči byl využíván nástroj Chrome DevTools[3]. Tyto velikosti byly porovnávány s jednotlivými velikostmi boxů v knihovně CSSBox. Na obrázku 7.1 si lze prohlédnout ukázku porovnání referenčního testu vůči knihovně CSSBox.

Obrázek 7.1: Na obrázku je vyobrazen referenční test ve webovém prohlížeči (nahoře) a jeho následné zobrazení pomocí knihovny CSSBox (dole).

Testovací sada byla postupně i rozšiřována o nové typy testů, aby bylo dosaženo lepší úspěšnosti a zároveň mohla být prováděna další korekce zdrojového kódu. Nové testy byly vybírány z již zmíněné testovací sady ze stránek konsorcia W3C. Bohužel z této sady nemohly být vybrány všechny testy, z důvodu, že je v nich využíván JavaScrpt pro testování výsledného obsahu. Toto omezení se dále shrnuto v kapitole 7.3. V tabulce 7.2 je zobrazena statistika výsledků již rozšířené testovací sady. Zdrojová data k této tabulce jsou součástí přiloženého CD v adresáři *web*.

Počet úspěšných testů	58
Počet neúspěšných testů	44
Celkem testů	102
Úspěšnost testů	56,862 %

Tabulka 7.1: Statistika výsledků automatického testování na základní testovací sadě.

Počet úspěšných testů	114
Počet neúspěšných testů	36
Celkem testů	150
Úspěšnost testů	76 %

Tabulka 7.2: Statistika výsledků rozšířené testovací sady po následné korekci zdrojového kódu.

[3] https://developers.google.com/web/tools/chrome-devtools/

7.2 Vlastní testovací stránky

Další forma testování byla prováděna pomocí vlastních webových stránek, které byly vytvořeny speciálně pro testování implementace algoritmu Grid Layout. Stránky byly v průběhu implementování postupně měněny. Díky těmto změnám bylo také odhaleno několik nedostatků v implementaci, na které bylo ihned reagováno následnou korekcí zdrojového kódu. Zde je výčet stránek, na nichž bylo testování prováděno:

- xnovak2b/grid
- xnovak2b/grid1
- xnovak2b/grid2
- xnovak2b/grid3
- xnovak2b/grid4

Na těchto stránkách byly testovány základní funkce. Jako je třeba umisťování itemů do explicitní i implicitní mřížky na základě jejich souřadnic nebo ověřování správného usazení automaticky umístěných itemů. První ze zmíněných stránek obsahuje několik příkladů, včetně jejich popisů, kde jsou hlavně testovány různé kombinace hodnot, které lze definovat jako velikosti stop mřížky. Předposlední stránka se zabývala zobrazením struktury, kterou lze i normálně použít pro webové stránky. Na poslední zmíněné stránce je vytvořen kalendář pomocí algoritmu Grid Layout. Ukázka tohoto kalendáře je znázorněna na obrázku 7.2.

Všechny zdrojové kódy těchto webových stránek jsou součástí přiloženého CD v adresáři *web*, rozděleny podle zmíněného seznamu.

Obrázek 7.2: Kalendář vytvořený pomocí algoritmu Grid Layout.

7.3 Shrnutí testování

Na základě testovaní pomocí testovací sady bylo objeveno několik nedostatků, které pomohli v korekci zdrojového kódu a jeho následného rozšíření o podporu nových prvků. Dle výsledků z tabulek 7.1 a 7.2 jde vidět, že postupná korekce kódu vedla ke zlepšení celkové úspěšnosti. Ve výsledku se dá říci, že zlepšení stouplo o 20 % vůči základnímu stavu, i při postupném přidávání nových testů.

Při testování byl zjištěn rovněž další nedostatek knihovny CSSBox, jedná se o nepodporování JavaScriptu. Tento nedostatek je docela velkým omezením v průběhu testování, protože většina testů je prováděna právě pomocí Javascriptu. Při spuštění takového testu knihovna CSSBox zobrazí pouze prázdnou stránku, a není tedy možné test vyhodnotit. Byly tedy vybírány jen testy, které knihovna CSSBox dokáže zobrazit.

Díky testování na vlastních webových stránkách bylo odhaleno nesprávné vykreslování itemů, pokud byly roztaženy přes více stop s různými velikostmi. Další nedostatek byl nalezen u itemů, které se rozkládaly mezi explicitní a implicitní mřížkou, šlo především o nesprávné přidělovaní jejich rozměrů. Oba tyto nedostatky byly následně odstraněny.

7.4 Další možnosti rozšíření

Tato práce umožňuje implementované řešení dále rozšiřovat a je také ve fázi kdy umožňuje různou optimalizaci při vývoji dalších částí. Grid Layout je stále vyvíjen, tedy je možné rozšířit tuto knihovnu o jeho nové prvky. Mezi tyto prvky lze zařadit tzv. subgridy[4], které umožňují vytvářet z itemů nové grid kontejnery. Další možnosti rozšíření o nové prvky Grid Layoutu je podpora inline gridu, tedy vlastnosti `display:inline-grid`, nebo přidání pojmenovaných oblastí v mřížce.

Jedním z důležitých rozšíření lze považovat i podporu JavaScriptu. Jak již bylo řečeno v kapitole 7.3, většina testů je prováděna právě pomocí JavaScriptu a není tedy možné provádět rozsáhlé testování.

Této knihovně by výrazně pomohlo i rozšíření o podporu protokolu HTTPS nebo přidat zpracovávání desetinného formátu u jednotek, aby desetinná část nebyla ořezávána.

[4] https://www.w3.org/TR/css-grid-2/

Kapitola 8

Závěr

Cílem této bakalářské práce bylo seznámit se s knihovnou CSSBox včetně její architektury. Prostudovat nový způsob rozložení webových stránek pomocí Grid Layoutu (mřížkového rozložení) a následně rozšířit knihovnu CSSBox o nové prvky tohoto rozložení.

Nejprve byly prostudovány technologie, které se ve spojení s webovými stránkami využívají, na to bylo navázáno studiem mřížkového rozložení a principu jeho fungování. Před samotnou úpravou knihovny CSSBox byla nejprve prostudována její aktuální architektura. Dále byla prostudována i knihovna jStyleParser, která je potřebná pro získávání potřebných stylů v knihovně CSSBox.

Na základě získaných znalostí byla nejdříve navrhnuta architektura systému Layout managerů, díky které je nyní možné přidělit jiný způsob rozložení jednotlivým boxům a zároveň umožňuje přidání nových způsobů rozložení. Tato architektura byla dále implementována aby bylo možné do knihovny CSSBox zavést již zmíněný Grid Layout a následně provést jeho implementaci. Provedená implementace byla průběžně testována na testovací sadě a vlastních testovacích stránkách.

Tato nová architektura včetně nového způsobu rozložení obsahu webových stránek se následně stane součástí oficiální verze knihovny CSSBox.

Literatura

[1] Andrew, R.: *Get Ready for CSS Grid Layout*. A Book Apart, 2016, ISBN 978-1-9375572-7-0.

[2] Burget, R.: CSSBox Manual. [Online; navštíveno 25.2.2019].
URL http://cssbox.sourceforge.net/manual/

[3] Burget, R.: jStyleParser Manual. [Online; navštíveno 27.2.2019].
URL http://cssbox.sourceforge.net/jstyleparser/manual.php

[4] Doležalová, M.: Layout stránky. [Online; navštíveno: 9.4.2014].
URL https://slideplayer.cz/slide/2453027/

[5] Gasston, P.: *Moderní web*. Computer press, 2015, ISBN 978-80-251-4345-2.

[6] Hissom, A. E.: History of HTML and CSS. [Online; navštíveno: 7.3.2019].
URL http://amyhissom.com/HTML5-CSS3/history.html

[7] Janovský, D.: Jak psát web: XHTML, jak se liší od HTML. [Online; navštíveno 28.2.2019].
URL https://www.jakpsatweb.cz/html/xhtml.html

[8] Meyer, A. E.; Weyl, E.: *CSS: The Definitive Guide*. O'Reilly Media, 2017, ISBN 978-1-449-39319-9.

[9] Michálek, M.: CSS Grid: Začíná druhé dějství layoutového převratu. [Online; navštíveno: 16.4.2019].
URL https://www.vzhurudolu.cz/prirucka/css-grid

[10] NOTHREM: GRID, aneb revoluce tabulkového layoutu. [Online; navštíveno: 20.3.2019].
URL http://css.chobits.ch/grid/

[11] Ondrák, L.: *Implementace flexibilního rozložení ve zobrazovacím stroji CSS*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2019.

[12] SaeidMarouski: The box model. [Online; navštíveno: 10.4.2019].
URL https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Box_model

[13] Snížek, M.: *CSS pro zelenáče*. Neocortex, 2004, ISBN 80-86330-14-1.

[14] Výuka informatiky: WebDesing - Layout stránky. [Online; navštíveno: 8.4.2019].
URL http://info.spsnome.cz/WebDesign/HTML-Layout

[15] Stohwasser, P.: pestujemeweb.cz: Co je CSS. [Online; navštíveno: 5.3.2019].
URL http://www.pestujemeweb.cz/obsah/css/co-je-css.php

[16] tvorba-webu.cz: DOM: Document Object Model. [Online; navštíveno: 2.3.2019].
URL https://www.tvorba-webu.cz/dom

[17] W3C: CSS Grid Layout Module Level 1. [Online; navštíveno: 15.4.2019].
URL https://www.w3.org/TR/css-grid-1/

[18] W3C: HTML & CSS. [Online; navštíveno: 4.3.2019].
URL https://www.w3.org/standards/webdesign/htmlcss

[19] W3C: What is the Document Object Model? [Online; navštíveno: 2.3.2019].
URL https://www.w3.org/TR/1998/WD-DOM-19980720/introduction.html

[20] W3Schools: Introduction to HTML. [Online; navštíveno 26.2.2019].
URL https://www.w3schools.com/html/html_intro.asp

Příloha A

Obsah CD

Na přiloženém CD lze nalézt tyto následující adresáře a soubory:

- *doc* – zdrojové kódy technické zprávy
- *CSSBox* – zdrojové kódy celé knihovny, včetně rozšíření
- *web* – zdrojové kódy vlastních testovacích stránek a excel soubor s výsledky
- *diff* – diff soubor vytvořeného rozšíření
- *readme* – informace a manuál k projektu ve formátu *Markdown*
- *xnovak2b-Grid-Layout.pdf* – technická zpráva ve formátu pdf

Příloha B

Manuál pro zprovoznění projektu

Tato bakalářská práce byla vyvíjena ve vývojovém prostředí IntelliJ IDEA, manuál tedy bude cílit pro toto prostředí.

Nejprve je potřeba naklonovat zdrojové kódy knihovny CSSBox a jStyleParser ze serveru *github.com* pomocí následujících příkazů:

```
git clone https://github.com/radkovo/CSSBox.git
git clone https://github.com/radkovo/jStyleParser.git
```

V prostředí IntelliJ zvolit *Import Project*, najít soubor `pom.xml` knihovny CSSBox a pomocí něj tento projekt naimportovat. V pravé části vývojového prostředí kliknout na záložku *Maven Projects*, kde je potřeba najít tlačítko *Add Maven Projects*. Poté je třeba naimportovat soubor `pom.xml` z knihovny jStyleParser. V poslední řadě je třeba kliknout na tlačítko *Generate Sources and Update Folders For All Projects*.

Nyní je vše připraveno a projekt lze spustit pomocí třídy `BoxBrowser` uložené v adresáři *demo*.